LA-UR- 10-08211

Approved for public release;
distribution is unlimited.

Title:	In-situ Sampling of a Large-Scale Particle Simulation for Interactive Visualization and Analysis
Author(s):	Jonathan L. Woodring 209118 CCS-7 James P. Ahrens 113788 CCS-7 Katrin Heitmann 175878 ISR-1
Intended for:	Eurographics/ IEEE-VGTC Symposium on Visualization 2011

Form 836 (7/06)

Eurographics/ IEEE-VGTC Symposium on Visualization 2011
H. Hauser, H. Pfister, and J. van Wijk
(Guest Editors)

Volume 30 (2011), Number 3

In-situ Sampling of a Large-Scale Particle Simulation for Interactive Visualization and Analysis

J. Woodring, J. Ahrens and K. Heitmann

Los Alamos National Laboratory, U.S.A.

Figure 1: *The image on the left is from a 256^3 (16 million) MC^3 dark matter cosmology data set. The image on the right is a 32 thousand random sample of that data.*

Abstract

We propose storing a random sampling of data from large scale particle simulations, such as the Roadrunner Universe MC^3 cosmological simulation, to be used for interactive post-analysis and visualization. Simulation data generation rates will continue to be far greater than storage bandwidth rates and other limiting technologies by many orders of magnitude. This implies that only a very small fraction of data generated by the simulation can ever be stored and subsequently post-analyzed. The limiting technology in this situation is analogous to the problem in many population surveys: there aren't enough human resources to query a large population. To cope with the lack of resources, statistical sampling techniques are used to create a representative data set of a large population. Mirroring that situation, we propose to store a simulation-time random sampling of the particle data to cope with the bottlenecks and support interactive, exploratory post-analysis. The particle samples are immediately stored in a level-of-detail format for post-visualization and analysis, which amortizes the cost of post-processing for interactive visualization. Additionally, we incorporate a system for recording and visualizing sample approximation error information for confidence and importance highlighting.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications— H.3.m [Information Storage and Retrieval]: Miscellaneous—

1. Introduction

Modern scientific discovery greatly depends upon computer modeling and simulations. Interactive, exploratory visualization has proved to be essential for scientific discovery and analysis [AHP*10]. For scientific simulations, leadership supercomputing is in the petascale range (10^{15} flops). It is expected in the next decade, supercomputers will move into the exascale range (10^{18} flops). The amount of data generated through simulations on these machines is a significant concern to the scientific simulation community, as the amount will surpass our capability to interactively explore and analyze it [JR07].

Storage read and write bandwidth on leadership supercomputing has not kept pace with computational speed [AHL*10], therefore it is not possible to store all generated data. Even if the storage bandwidth existed, the storage capacity would fill up immediately. Hence, analysis accuracy suffers due to only being able to store a small fraction of the original data. For example, the RoadRunner Universe MC^3 [HPL*09] is a large N-body cosmology simulation of dark matter physics. An MC^3 time step of 4000^3 (64 billion) particles with 36 bytes per particle takes 2.3 TB per time slice. A Panasas parallel filesystem [pan] connected to Los Alamos National Laboratory (LANL) simulation supercomputers can optimistically operate at 10 GB/s. At a very optimistic peak, it takes 4 minutes to move the 4000^3 particle data to or from storage in parallel. Under realistic operating conditions, it takes approximately 30 minutes (or longer depending on filesystem utilization) to write the data from the supercomputer to storage. Very few time steps are ever written to disk due to the storage bandwidth imbalance [JR07].

Displays have not kept pace with data sizes and ultimately they have an upper bound based on visual capability [Dee98]. Displays are in the megascale while data is in the gigascale and terascale per time slice. Even with 256^3 particles (16 million), a small data set in comparison, it is impossible to see details and structure in visualization of the particles, as seen in Figure 1. Therefore, it is not possible to see all the data from a large-scale simulation even if it was possible to store and retrieve it.

The contribution of this application paper is an *in-situ*[†] data sampling solution that stores particle data from simulations for later interactive post-analysis and visualization. Our method overcomes the bottlenecks by reducing data through *in-situ* sampling directly from the simulation. Sampling at run-time also allows us to quantify the amount of error we introduce into the data and subsequently visualize this error at analysis time. We use statistical random sampling methods [Loh10] to create an unbiased data representation primarily for scientific analysis, but can be used for visualization, as well. The samples are encoded in a visually coherent LOD format directly from the simulation, to amortize the cost of LOD encoding in post-processing, which allows for immediate interactive visualization. We apply our technique to the data from the RoadRunner Universe simulation and demonstrate its applicability to understanding these scientific simulation results.

2. Methodology and Related Work

We describe our methodology for interactive, exploratory post-analysis and visualization of large-scale particle data simulations, outlined in Figure 2. Simulations are limited by storage bandwidth and can only store a small fraction of data compared to the entirety of the original data (conservatively estimated under 10%) [JR07, AHL*10, SBH*10]. As current trends suggest, this fraction will continue to shrink as storage technology is not able to keep pace with supercomputing growth. The standard methods to cope with the lack of I/O bandwidth at simulation run-time are to store very few time steps, drop data dimensions (variables), drop precision (converting doubles to floats), and/or use *in-situ* visualization [TYR*06, MFMe10]. When doing *in-situ* visualization, there is a trade-off of what data to save for later analysis and what operations can be performed afterward [TCM10]. Our goal is to sample the entire data set so that, for later analysis, nearly all operations are available for interactive exploration.

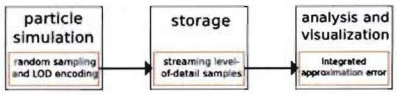

Figure 2: *The end-to-end pipeline from data generation to exploratory post-analysis of large-scale particle data.*

In the next section, we explain our random sampling method [Loh10] for particle data across space and/or data dimensions to support exploratory post-analysis and visualization. Random sampling of particles allows us to spread information loss [CJ10] across other dimensions (space and data) and store the data in a "nearly raw" format. This is in contrast to focusing the primary amount of information loss in the temporal dimension and/or precision.

Another benefit of sampling particle data at simulation-time is that we are able to record the *approximation or estimation error* of the sample data compared to the original data, described in Section 5. Each particle in our output sample (or a particle in a LOD visualization) represents missing particles that are not stored (or displayed). We define *approximation or estimation error* to be how accurately (or inaccurately) the sample data represents the original data. We are able to visually highlight the particle samples with high approximation error, described in Section 6.1.

It is standard practice in visualization to reorganize and post-process data on storage to support interactive visualization through streaming, level-of-detail (LOD), and prioritization [FSW09, AWD*09, HE03]. LOD visualization is necessary to allow data to scale under the run-time bottlenecks in storage, networks, and displays. For example in massively parallel visualization systems, the bulk of the time is spent in I/O and it makes time-varying visualization a non-interactive, batch process, along with the data size being larger than the number of available pixels. Therefore, we adopt a streaming LOD visualization methodology for our sample particle data, described in Section 6.

Though, the practice of LOD post-processing will not scale due to the storage bottleneck, and we believe it has al-

[†] In-situ refers to running analysis, visualization, or co-computation as part of the simulation code while it is executing in a batch on the supercomputer.

ready gotten to the point of inaccessibility for scientists and the scientific workflow. The authors of [FSW09] cite that it only took 5 hours to process their data for visualization, but that is 5 hours of lag time added between the simulation and analysis steps. Additionally, computational power is growing faster than storage bandwidth, and in the limit, there will only be one time slice or a few variables that will be stored (at the end of the simulation). LOD post-processing cannot recover the lost temporal or field data that is never stored.

Our solution is to store the LOD format directly from the simulation, described in Section 4. Our process removes the lag time through amortizing the cost of LOD encoding at simulation time. Considering that the storage is the slowest component of computing, we remove one read and one write of the data set between simulation and analysis, by moving the LOD encoding into the simulation. Removing the lag time between simulation and visualization provides a faster scientific workflow turnaround time, increasing cosmologist productivity.

In Figure 3, we show a cosmology dark matter data set from MC^3 [HPL*09] visualized with a streaming LOD framework in ParaView and VTK [AWD*09] utilizing our LOD sampling methodology. In this work, we are primarily concerned with the data management and representation of the data for scientific analysis operations. There are more sophisticated visualization methods for rendering point data [FSW09, HE03], and our representation and visualization methods may not be ideal. Rather than visual accuracy, we focus on large-scale data management and scientific analysis [JS03] in Section 7.

Finally, we measure the performance of our method in Section 8. We assume that our LOD data format will logically layout the data for efficient linear disk access at analysis time, as with many other LOD systems [FSW09, AWD*09, HE03]. The primary performance metric we measure is the additional *in-situ* overhead for sampling and LOD encoding of MC^3 particle data.

3. Stratified Random Sampling of Particle Data

We generate an *in-situ* random sample from the original MC^3 particle data during the simulation by sampling from spatially contiguous blocks of particles. Each block or stratum has an equal number of particles that are allocated proportional to the number of particles in a block. This sampling strategy is a *stratified random sampling* (SRS) [Loh10]: each stratum is the same size, the number of sample points taken from each stratum is proportional, and the inclusion probability of every particle in the final sample is equal.

There are several reasons that we utilize SRS for particle sampling: Stratified random sampling protects us from acquiring a bad sample [Loh10]. By spreading the sample points across spatial strata, we ensure that there is a good sampling spread. Also, a stratified sample data often provides more precise (lower variance) estimates of the original data. Usually, simulations are run such that the data is simulated in a spatially contiguous region per processing element. Therefore, the data is naturally separated into large spatial strata. The MC^3 cosmological simulation, which we use in our application and testing, simulates the dark matter particles in this manner. Finally, separating the particle data into spatially contiguous blocks naturally fits into a streaming LOD visualization schema [AWD*09] and allows us to extend our sampling method into a stored LOD format. The LOD encoding will be described later in Section 4.

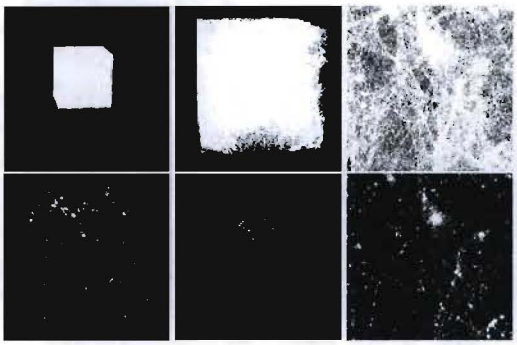

Figure 3: *An example of a visualization of a particle cosmological data set under zooming. The images on the top are the data at full resolution. The images on the bottom use samples from an* in-situ *generated LOD hierarchy.*

3.1. Implementing SRS with kd-tree Sorting

We assume that the particle simulation is pre-partitioned per processor into large spatial blocks (strata), as is the case with the MC^3 simulation. To acquire a sample size of s from N particles, we allocate a proportional number of samples per processor block. Assuming there are are p_i particles on processor i, block i is allocated $p_i \cdot s/N$ random samples. The total sample is the union of all of the random samples taken from every block (stratum).

We utilize a kd-tree per processor to recursively divide the particle data into smaller stratum. The kd-tree sorts the particle data by spatial axes to divide the data into equal density spatial partitions (strata) for stratified random sampling (SRS). The smaller stratum ensures a more even spread of random samples across space (and it is somewhat similar to signal anti-aliasing by putting more samples in high frequency areas). Furthermore, we will use this kd-tree sampling construction to create a level-of-detail (LOD) structure for interactive post-visualization, explained later in Section 4.

Figure 4 shows how particle data is sorted into spatial strata and one random sample is taken per stratum. Our SRS algorithm is the time that it takes to construct a parallel kd-tree or $O((N/p)log(N/p))$, where N is the total number of particles and p is the number of processors. The pseudocode

to recursively create a SRS of parallel particle data with a kd-tree per processing element follows:

```
s = "sample size" < N
N = "total number of particles"

# get local particles, sample the data
# with a fraction of the number of particles, and
# then gather the samples across all processors
p = get-particles()
gather(kd-tree-random-sample(p, s * |p| / N))

# recursive sampling function using a kd-tree.
# part is the particles and ss is the sample size
function kd-tree-random-sample(part, ss):
  # if the sample size is one, get a random sample
  # from the kd-tree leaf (block/stratum)
  if ss == 1:
    return one-random-sample(part)
  # else split the particle data into two halves
  # and sample on each half, joining the samples
  else:
    left, right = median-sort(part, axis(d))
    return append(kd-tree-sample(left, ss/2),
                  kd-tree-sample(right, ss/2))
```

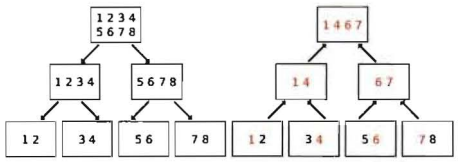

Figure 4: *The left image shows 1D particle data being sorted into a kd-tree. The right image shows the generation of a four particle random sample of the data. One random sample is taken per stratum (leaf nodes) (red numbers show the random sample) and joined into one sample.*

4. Level-of-Detail Construction

We extend our kd-tree stratified random sampling (SRS) algorithm, presented in the previous section, to construct a streaming level-of-detail (LOD) format for particle simulations. This algorithm (the sampling process and LOD generation) is done *in-situ*, such that the sampling and simulation storage methods are integrated into one algorithm, therefore the time taken is $O((N/p)log(N/p))$. Assuming that the particle data is stored in a linear array, and the array can be sorted, the kd-tree can be constructed in place with no extra memory (median sort/nth element). Otherwise, we can store an array of indices to mark how to restore the previous order of the data after kd-tree sorting (or copy the data if there is enough memory).

Our format is similar to other tree based LOD structures with spatial blocks of sample particles laid out linearly on storage [FSW09, AWD*09, HE03]. At each level of the hierarchy, we store spatially contiguous blocks of particles which represent a lower resolution SRS of the stored particle data in an LOD subtree. Each block in the LOD hierarchy has a number of particles equal to the sample size s and the number of blocks per level is 2^v where v is the level ($v \geq 0$). For simplicity, we assume the number of processors are a power of 2 and the sample size is a power of 2.

Figure 5 shows a diagram representation for creating the LOD hierarchy per processor. Figure 6 shows the completion (reduction) of the LOD hierarchy from per processor samples. In the following, we show the modification of the earlier pseudocode to support SRS and LOD storage of the particles. The main differences are that the LOD levels are written to disk as they are generated, the data can be written out in full resolution in the LOD structure, and that the LOD structure is built and completed in parallel across processors (assuming that each processor has a spatial block of particles, as is in the case with MC^3).

```
s = "sample size" < N
V = "set of LOD levels to store"
f = "LOD level to store full resolution data" >= -1

# get local particles and sample the data
# generate the LOD tree per processor
p = get-particles()
sample = sample-and-lod(p, s, 0)

# reduce the LOD tree from the per processor LOD trees
for d from -1 to -log2(number-of-processors()) by -1:
  # sibling is the opposite processor in a spatial
  # ordering of processors in simulation space
  sibling = get-opposite-sibling-rank(d)
  if get-rank() < sibling:
    send(sibling, sample)
    break
  else:
    if d is in V:
      sample =
        sieve-and-store
          (append(sample, receive(sibling)), d)

# recursive sampling and LOD construction
# part is the particles, ss is the sample size,
# and d is the current kd-tree depth
function sample-and-lod(part, ss, d):
  # write the full resolution data as a LOD level
  # if it is enabled (not -1)
  if d == f:
    write-to-storage(part)
  # if the sample size is one, acquire a random sample
  # from the kd-tree leaf (block/stratum)
  if ss == 1:
    return one-random-sample(part)
  # else split the particle data into two halves
  # and build the LOD tree on both halves
  else:
    left, right = median-sort(part, axis(d))
    if d < max(V):
      sample = append(sample-and-lod(left, ss, d + 1),
                      sample-and-lod(right, ss, d + 1))
    else:
      sample = append(sample-and-lod(left, ss/2, d + 1),
                      sample-and-lod(right, ss/2, d + 1))
    # if it a LOD level filter the samples
    if d is in V:
      return sieve-and-store(sample, d)
    # else return the samples to the parent
    else:
      return sample

# build a random sample to pass to a parent node
# in the LOD tree and store this sample
function sieve-and-store(sample, d):
  # every k sequential particles in the list are
  # from kd-tree siblings: randomly pick one of the
  # k particles to go to the parent node where
```

```
# k = 2^(last stored LOD level - this LOD level)
k = 2^(get-last-stored-level(V, d) - d)
for i from 0 to |sample| by k:
  pick = random(k)
  for j from 0 to k by 1:
    if j == pick:
      parent = append(parent, sample[i+j])
    else:
      children = append(children, sample[i+j])
  write-to-storage
    (append(parent, random-permutation(children))
  return parent
```

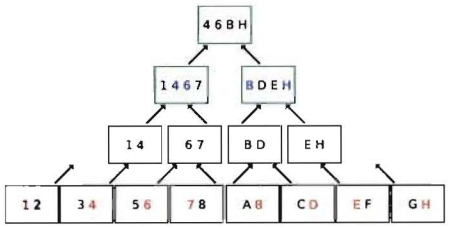

Figure 5: *Per processor creation of a two level LOD hierarchy with sample size of four. The bottom of the tree are the kd-tree leaves and red numbers are the random samples from the leaves. The green nodes are the stored LOD levels. The blue numbers are the random samples (random particle out of every two) propagated to the parent levels.*

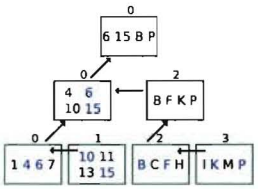

Figure 6: *Completion (reduction) of a LOD hierarchy with sample size of four. The green nodes at the bottom of the tree are the samples acquired per processor. Blue numbers are the random samples (random sample of every two particles) propagated to the parent levels.*

4.1. LOD Storage Size

Selected levels in the LOD hierarchy are written to storage. The number of particles stored per LOD level is $2^v \cdot s$ where v is a level in the LOD hierarchy ($v \geq 0$) and s is the sample size ($1 \leq s \leq N/2$). Given V ($V \subset Natural$), a set of LOD levels to store, the total number of particles written to storage will be $\sum_{v \in V} 2^v \cdot s$. This sum will be less than or equal to the total number of particles N, assuming that $\forall v \in V : 2^v \cdot s \leq N/2$ (the highest resolution LOD level is half the original data).

This is because the highest possible resolution of a level, which is not the full resolution data, is an SRS such that the strata are two particles each (every other particle is randomly sampled). The next possible highest resolution data will be $1/4$ of the original data, the next will be $1/8$, etc. (lower resolution levels are at a maximum a $N/2$ SRS given the algorithm in Section 4). Given this geometric series and the Equation 1, the LOD tree will at most be N of the original data, where $degree = 2$ (the kd-tree degree). Therefore, the size of the data is $N/(degree-1) = N/(2-1) = N/1 = N$. Thus the maximum amount of data stored in an LOD tree is N, not including the full resolution data.

$$\begin{aligned} \sum_{k=1}^{depth} \frac{N}{degree^k} &= N \frac{degree^{depth} - 1}{degree^{depth}(degree-1)} \\ \sum_{k=1}^{\infty} \frac{N}{degree^k} &= \frac{N}{degree - 1} \end{aligned} \tag{1}$$

N or $2N$ (adding the full resolution data) is potentially too large for our use case because of storage bandwidth limitations (unless we are only concerned with amortizing the cost of building the LOD structure). We can scale the size of the stored data *in-situ* and significantly reduce the amount of written data to fit within the storage bandwidth. We reduce the size by storing a few selected levels of the LOD hierarchy, limiting the depth of LOD hierarchy, and/or not storing the full resolution data. By not storing the full resolution data, it immediately reduces the data size by N.

For example if we store every third level in the kd-tree, starting with the level that is $1/8$ of the data, at most the stored data will be $N/7$ of the original data set. Every level will be $1/8$ of the previous level (this follows from the construction algorithm if every third level is stored). Thus, $N/8 = N/degree$, $degree = 8$ and $N/(degree-1) = N/(8-1) = N/7$. In another example, if we only store data starting from root (zeroth) and third levels of the kd-tree, the data size will be $2^0 * s + 2^3 * s = 9 * s$, where s is the sample size, which can be significantly less than N depending on the size of s.

In Section 8, we measure the storage size and timings for different configurations of LOD hierarchies. In Section 7, we perform scientific analyses, which are important to the cosmology domain on sampled MC^3 particle data, and compare sample data to analysis with full resolution data.

4.2. LOD Visual Continuity

In our construction algorithm of the level-of-detail (LOD) hierarchy, every LOD level is a a stratified random sample (SRS) of the original data, but each level is also a subset of higher resolution levels; i.e., LOD levels are not completely independent random samples. We preserve this subsetting property so that there is visual continuity during visualization between LOD levels: particles do not pop in or out ("sparkle") on resolution changes. By transitivity, if $A \subset B$ and $B \subset C$, then $A \subset C$, which means that by moving from any LOD level A to C, A will always be visually depicted in C. Figure 7 shows an abstract depiction of the visual continuity between LOD levels.

This follows from our algorithm in Section 4 such that

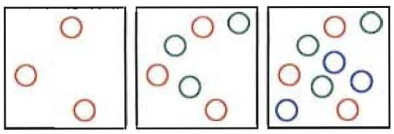

Figure 7: *An abstract depiction of LOD particle data under increasing resolution with visual continuity. The particles in the lower resolution data are present in the higher resolution data.*

each block in the LOD structure is the combination (reduction) of two higher resolution blocks. We create a lower resolution block through using SRS on the higher resolution blocks (randomly picking a particle from every k particles, or one random sample from every stratum of size k).

Through the recursive algorithm, a node or block at level L is the combination of two or more nodes from level $L+M$ where $M \geq 1$. The new particle sample is created by randomly taking one out of every 2^M particles (ordered list of particles by kd-tree sorting), as seen in Figure 5 and 6. Thus, the root node is a SRS of any level in the LOD hierarchy. Also, any LOD node is an SRS of its subtree. Therefore, every level and every node in the hierarchy is a subset of all higher resolution levels and blocks.

4.2.1. Smooth Visual Continuity

Additionally, we add an additional feature to the LOD storage for smooth visual continuity between displayed levels. If we were to switch resolutions between stored levels, the new amount of particles would, at a minimum, always be half or double the number of visible particles creating a large visual discontinuity. We remove the popping by smoothing the number of visible particles through "randomly" adding or removing particles one by one.

We achieve this by the way the particles are ordered, stored, and selected. In our algorithm in Section 4, the combined particles from child nodes to create a new node in the LOD hierarchy are divided into two partitions: particles that are filtered up to a parent node and particles that are not (the particles which are only in a child node relative to a parent). The particles which do not go to the parent block are written to storage in a randomly permuted order.

During visualization and analysis, an LOD level, depending on zoom factor, can be partially rendered to simulate unstored, finer-grain LOD levels. This is done by incrementally adding or removing particles from the randomly ordered list, one by one. Once we have added or removed all of the particles from the list, we have added all of the particles that overlap with children nodes, or conversely, we have removed all of the particles until only those, which overlap with the parent node, remain. We then seamlessly jump to the next stored LOD level without visual popping because the displayed particle set either equals the particle set of the parent node or the maximum set of particles that overlaps with the child nodes. Figure 8 shows the particle organization of blocks on storage. The bottom half of Figure 3 is an example of the particle data under zooming and incrementally adding more particles.

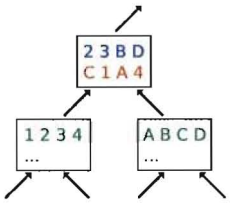

Figure 8: *The top node is a parent block created from the green particles selected from the two child nodes. The blue particles are samples randomly selected to go to the parent node. The red particles are the unselected particles that overlap with children. To render a partial LOD level (between the top and bottom nodes), we render all of the blue particles and and incrementally add red particles.*

5. Approximation Error Measurement

In the LOD algorithm in Section 4, the particles are sorted into spatial blocks or strata. One random particle is acquired per stratum. The aggregate sample is representative of the whole population and samples in a localized region are representative of the local area. While the aggregate may be a good representative, the problem is that a single sample can be a poor representative of the stratum that it is taken from.

Through our sampling and LOD visualization, there is a loss of transmitted information from the simulation to the cosmologist [CJ10]. This has been the case in large-scale simulations, as the storage bandwidth is not large enough to accommodate and transmit all of the information from the simulation to storage [JR07, AHL*10]. Our stratified random sampling (SRS) introduces *approximation error* into the representative data, as is true with many other level-of-detail (LOD) and data transformation methods. We define *approximation or estimation error* to be how accurately (or inaccurately) our sample data represents the original data.

The way we are able to overcome some of the information loss is that we are able to measure and quantify information about the original data to augment the sample data *in-situ*. This additional information can be used to quantify the the quality of the sample data, measure the amount of introduced error compared to the original data, and visualize this information. This provides a data quality assurance as we are are able to directly query the generated source data.

The measures that we utilize for MC^3 are local statistical properties on the particle fields (variables). The particle data contains position (x, y, z), velocity (vx, vy, vz), and mass. We measure the sample mean, variance, and sample size ($\bar{x}$, s^2, and n) of the the data within each kd-tree leaf (stratum). The sample mean and variance are important measures because of their relationship to signal-to-noise ratio and the

coefficient of variation [Loh10]. These statistical properties are assigned to the sample particle taken from each strata to represent the original data. The sample mean, variance, and size can be propagated up the LOD hierarchy when blocks are combined together:

$$\begin{aligned} \bar{x} &= \frac{n_1\bar{x}_1 + n_2\bar{x}_2}{n_1+n_2} \\ s^2 &= \frac{n_1^2s_1^2 + n_2^2s_2^2 - n_1s_2^2 - n_2s_1^2 - n_1s_1^2 - n_2s_2^2 + n_1n_2s_1^2 + n_1n_2s_2^2 + n_1n_2(\bar{x}_1-\bar{x}_2)^2}{(n_1+n_2-1)(n_1+n_2)} \\ n &= n_1+n_2 \end{aligned} \quad (2)$$

Therefore at every LOD level, each sample particle will have the local statistical properties of the original strata it represents. In Section 6.1, we will discuss methods for utilizing the local statistical properties and visualizing the approximation error (comparing the sample data against the original data).

If we calculate the statistical metrics top-down, they give us a mechanism to sample more (add additional LOD levels) in high variance areas or the where samples do not mirror the sample mean (assuming the LOD hierarchy does not store highest resolution data). This modification of allows the kd-tree to grow unevenly and will use more storage bandwidth than a static amount. It is difficult to predict how much additional bandwidth this can use, as it is entirely data dependent.

6. Streaming LOD Analysis and Visualization

Figure 9: *Cosmology data being incrementally updated over time in ParaView.*

To visualize our level-of-detail (LOD) MC^3 data, we utilize a prioritized, incremental visualization and analysis system. We have modified streaming ParaView and VTK to handle resolution requests for pieces (blocks) [AWD*09]. For a given view, pieces that lie outside the view are culled. Within the view, pieces are ordered, filtered, and incrementally rendered in priority order depending on the requested resolution, as seen in Figure 9. We acknowledge that our particle rendering may be considered primitive compared to other techniques [FSW09, HE03]. Even though our primary goal was an end-to-end data management system for scientific analysis of MC^3, nothing precludes integrating a more sophisticated rendering method in the future.

"Qualitative" interactive visualization is an important goal for insight, but the scientific workflow and discovery is completed with "quantitative" scientific analysis [AHP*10]. This has been one of our main motivating goals for storing the MC^3 data in a "nearly raw" format, rather than being a "visualization optimized" format: the data can be utilized in an existing visualization system combined with quantitative scientific analysis to enable the cosmology scientists. We can directly leverage many of the scientific analysis tools directly on the data without additional transformation, such as histograms, thresholding, or specific cosmology filters developed for MC^3 like FOF halo finding [WHA*10] in ParaView and VTK. Also, the cosmologists can use their own analysis tools directly on the sample data, as the sample data from an LOD level can be concatenated together into their original "cosmo" format (each block or level in our LOD hierarchy is the "cosmo" format).

6.1. Visualization of Approximation Error

Each sample particle contains local statistical properties of the stratum it represents. The local statistical properties we record are the sample mean, variance, and size. There are two main ways that we can visually utilize this information: highlight areas that have high variance in the original data and highlight areas where the statistics (mean) differs from the original data. In both these cases, the visualizations are a mechanism to direct the cosmologist to parts of a visualization that may be under-represented and/or the results may not be as accurate as necessary.

In the first case where we highlight areas that have high variance, this directs the cosmologist to potential interesting areas of the data set. When viewing low-resolution LOD data, the cosmologist should zoom into the highlighted area, because individual particles under-represent the original data in the current visualization (there is a large spread of values in the original data and a single sample is a poor representative). Figure 10 shows the identification of a halo (cluster of dark matter particles) in a low density region, at low resolution, via velocity variance color highlighting.

Figure 10: *A halo in a low density region is visually located in a low resolution sample by velocity variance color highlighting. LOD resolution increases left to right with zooming.*

In the second case, we compare local statistical properties (mean) in the sample data against the mean of the original data. If the mean varies significantly, we can highlight those areas to show that the sampling at that LOD level may not be a good representative for a particular LOD level. As before, the cosmologist should zoom into a higher resolution sample before doing any calculations on the data in that localized area.

7. Comparison of Sample Data

We compare the accuracy of a low resolution MC^3 sample data in statistical measures and data value variation across space in the velocity field. Figure 11 shows several histograms of the velocity components (top row) and the velocity component variation across space (bottom row). The red line is a 32768 sample (.19%) of a 256^3 particle data and the black line is the value of the original data. As we can see in the histograms, the sample data almost exactly represents the histogram of the velocity components (the black curve overlaps the red curve). The velocity, as it varies across space, nearly matches the original data, as well, except in the velocity x component, where it varies slightly (the red curve overlaps the black curve).

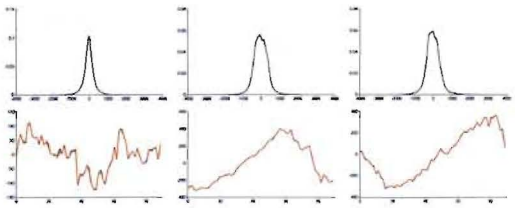

Figure 11: *Red curve is a 32768 sample of a 256^3 MC^3 particle data set (black curve). Top row is the histogram of the velocity components (vx, vy, vz, left to right). The red curve occludes the black curve. Bottom row is the average value of the velocity component across space (vx in x, vy in y, vz in z, left to right). The black curve is occludes the red curve.*

Secondly, we measure the spatial sampling accuracy by using a spatial feature finding and statistical measure on that feature. In cosmological science, halos are an important feature in dark matter simulations, which is a clustering of dark matter particles in the MC^3 data. An important measure of the halos is the "mass function", a histogram of halo masses [AHP*10]. In Figure 12, we show the comparison of running a halo finding algorithm and the subsequent mass function on different sample sizes of a 256^3 particle data set. The smallest sample size .19% of the data is not able to replicate the mass function. On the other hand, the samples of 1.6% and 12.5% are able to approximate the full resolution mass function.

8. Resource Timings

A RoadRunner Universe MC^3 2048^3 particle run generates 8 billion particles per time slice. A time slice is 309 GB total at

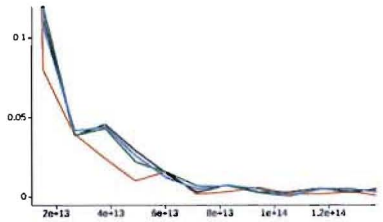

Figure 12: *The mass function for different sample sizes of 256^3 particles. The black curve is the original data. The red, green, and blue curves are .19%, 1.6%, and 12.5% samples, respectively.*

36 bytes per particle. In a 512 way run, the per process files are approximately 604 MB each. The data is stored from the simulation on a Panasas file system [pan] connected to Cerrillos, a hybrid 4 x AMD/4 x Cell supercomputer with 16 GB each, featuring IBM QS22 and LS11 blades similar to RoadRunner. The nodes are connected via 4 x DDR Voltaire Infiniband in a fat-tree topology.

For the sampling and LOD, only a block size (sample size) extra memory is required as the particles can be kd-tree sorted in place with STL nth_element. On the processor reduction phase, we use an optimization over the algorithm in Section 4 such that the particles are sampled per processor before merging (reduction). Then, the particles are gathered to the write processor for LOD block storage. This requires only an extra block size of memory to gather a sample block across 2^d processors, where d is the largest level gap in the across-processor gather hierarchy.

In the following, we time the creation for sampling and storage of an LOD structure of 2048^3 MC^3 particles. Table 1 shows the different configurations using a 32768 sample size per block (~1.1 MB per block). The first row is writing the full resolution data and a complete LOD hierarchy. The second row is writing the complete LOD hierarchy without the full resolution data. Third through seventh rows are writing selected levels out of the LOD hierarchy. The last row is the original full resolution data storage write method from MC^3. We can see that the largest impact for an LOD data size is dependent on the highest resolution level.

write config	size	max res	total data
full LOD + full res	618 GB	N	2N
full LOD	309 GB	N/2	N
every 2nd level LOD	103 GB	N/4	~.333N
every 3rd level LOD	44 GB	N/8	~.142N
every 4th level LOD	82 GB	N/4	~.265N
every 5th level LOD	39 GB	N/8	~.126N
every 6th level LOD	4.9 GB	N/64	~.016N
write full res	309 GB	N	N

Table 1: *Different data sizes for different LOD configurations with a 32768 sample size per block. The bottom row is the normal full resolution write of the "cosmo" format without sampling and LOD storage.*

Table 2 shows the amount of time savings for the cosmologist between simulation and interactive analysis. It takes 210s to write the data in its original full resolution "cosmo" format from MC^3 and 153s to read that same data, both 512 way from the Panasas. Therefore, we have a savings of 364s (~6 minutes) per time slice between simulation and analysis by performing the LOD encoding at simulation time (removal of one extra read and write of the data). If we only consider the amount of time saved in post-processing, we still save 153s (~2.5 minutes) per time slice (removal of the extra read). There is the also beneficial factor that the data is immediately ready for interactive analysis in ParaView, removing one step from the scientific workflow process (removing post-processing).

write full	read full	lag save	post-proc. save
210s / slice	153s / slice	364s / slice	153s / slice

Table 2: *Original full resolution MC^3 data read and write timings. "Lag save" is the lag time savings between simulation and post-analysis per time slice (one read and write) through* in-situ *processing. "Post-proc. save" is the time savings per time slice if only post-processing is considered (one read).*

In Figure 13, we show the actual time taken to write the data in different LOD configurations. The right most bar shows the original write time per time slice. The first two bars show that encoding a full LOD structure (every LOD level) with or without a full resolution level does not save any simulation time. Though, there still is a total end-to-end time savings if you consider the time simulation time + analysis lag time of 364s / slice (Table 2). Sampling and writing the data into an LOD structure (2nd bar) has an overhead of 51s (approximately 18s to 24s is due to kd-tree sorting overhead of 256^3 (16 million) particles per processor).

Actual simulation time savings comes from sampling and writing less data, which can be seen in Figures 13 and 14. Beginning with the 3rd bar from the left (writing an LOD structure using every 2nd level, with a highest resolution of N/4 with N/3 data in total), we save 118 seconds (2 minutes) per time slice. In the extreme case where we are only writing ~1.5% of the original size, we save 192s (3.2 minutes) per time slice. The time taken is nearly correlated by the amount of data written. Though, "every 5th level stored" takes 8 seconds longer than "every 3rd level stored", even though the former stores less data. This is likely due to system noise and utilization, since Cerrillos is an actively used supercomputer and the Panasas file system is shared across several other supercomputers.

In Figure 15, we can see the amount of additional time slices that can be stored to increase time fidelity. If we sample the data, we can expect to be able to add approximately one to three additional time slices. This allows the cosmologists to make a trade-off between temporal and spatial fidelity using the same I/O time, which does not preclude writing out full resolution slices. For example, normally if we

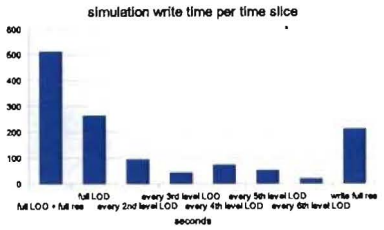

Figure 13: *Time taken to write sample and write the MC^3 particle data* in-situ *per time slice. The right-most bar is the original write time.*

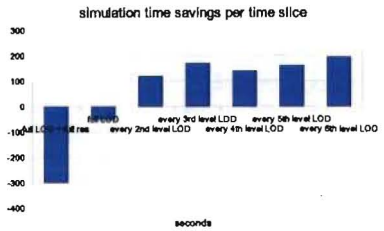

Figure 14: *Simulation time savings per time slice in different LOD configurations.*

normally write 100 full resolution time steps, we can trade half of those for a sampled version (for example, every other full resolution time slice) to add between 100 to 200 additional time slices (or potentially many more if using smaller resolutions). The remaining 50 could then be converted to 20 full LOD and full resolution time slices, for a mix of full resolution LOD data and low resolution LOD data.

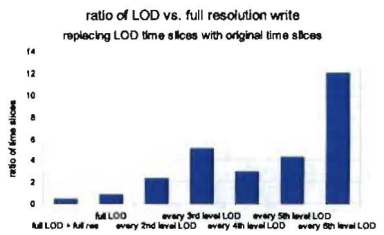

Figure 15: *Ratio of of our LOD write method time compared to the original write method time.*

Finally, we measure the performance of I/O read times in a streaming LOD system for different sample sizes in Figure 16. The peak read time from the disk we used is approximately 110 MB/s. In Section 4.2.1, we described a method for partially using blocks for intermediate LOD levels. As we can see, the larger block size (approximately 262 thousand samples or 8MB per block) has the best performance for full and partial I/O reads.

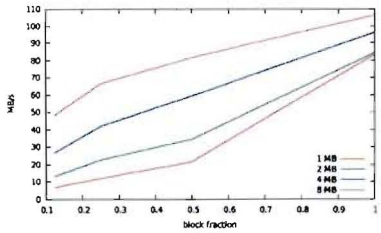

Figure 16: *Read block (sample size) I/O times, which vary based on partial read (striding) from a single mechanical disk. Peak throughput on a linear read is approximately 110MB/s on this disk.*

9. Conclusion

We have presented our method for *in-situ* stratified random sampling of the RoadRunner Universe MC^3 for post-analysis and visualization. We believe that this methodology opens a new *in-situ* venue to enable large-scale scientific analysis at extreme scales. We have only touched the tip of the iceberg as there are many different points of research for supporting interactive post-analysis at exascale: enabling other large-scale simulations, enabling different data types and sampling methods (such as *in-situ* wavelets on structured data), utilization of approximation error in analysis operations and propagation of error, integration into existing *in-situ* and co-processing frameworks and/or using I/O function call hooks, acquiring better statistical and summary information of the original data, and using sampling and dimensionality reduction on data axes other than the spatial axes.

In particular for MC^3, we need to look into measuring temporal error and ensuring temporal continuity, similar to LOD continuity. Halo tracking and merging over time are an important research area for the cosmologists. Secondly, outlier information and visualization is important for verification and debugging of simulation codes [AHP*10]. We would like to look into sampling the data axes (variables) and biasing the sampling towards saving outlier data. Finally, we would like to look into generating particle data at analysis time from original data statistical measures to increase the accuracy of halo finding for low resolution data.

References

[AHL*10] AHRENS J., HENDRICKSON B., LONG G., MILLER S., ROSS R., WILLIAMS D.: *Data Intensive Science in the Department of Energy*. Tech. Rep. LA-UR-10-07088, Los Alamos National Laboratory, Oct. 2010. 2, 6

[AHP*10] AHRENS J., HEITMANN K., PETERSEN M., WOODRING J., WILLIAMS S., FASEL P., AHRENS C., HSU C., GEVECI B.: Verifying scientific simulations via comparative and quantitative visualization. *IEEE Computer Graphics and Applications 30*, 6 (2010), 16–28. 1, 7, 8, 10

[AWD*09] AHRENS J. P., WOODRING J., DEMARLE D. E., PATCHETT J., MALTRUD M.: Interactive remote Large-Scale data visualization via prioritized Multi-Resolution streaming. pp. 1–10. 2, 3, 4, 7

[CJ10] CHEN M., JAENICKE H.: An information-theoretic framework for visualization. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (2010), 1206–1215. 2, 6

[Dee98] DEERING M. F.: The limits of human vision. In *2nd International Immersive Projection Technology Workshop* (1998). 2

[FSW09] FRAEDRICH R., SCHNEIDER J., WESTERMANN R.: Exploring the millennium run - scalable rendering of large-scale cosmological datasets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1251–1258. 2, 3, 4, 7

[HE03] HOPF M., ERTL T.: Hierarchical splatting of scattered data. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 57. 2, 3, 4, 7

[HPL*09] HABIB S., POPE A., LUKIĂĞ Z., DANIEL D., A ND N. DESAI P. F., HEITMANN K., HSU C. H., ANKENY L., MARK G., ET AL.: Hybrid petacomputing meets cosmology: The roadrunner universe p roject. vol. 180, p. 012019. 2, 3

[JR07] JOHNSON C., ROSS R.: *Visualization and Knowledge Discovery: Report from the DOE/ASCR Workshop on Visual Analysis and Data Exploration at Extreme Scale*. Tech. rep., Department of Energy Office of Science ASCR, Oct. 2007. 1, 2, 6

[JS03] JOHNSON C., SANDERSON A.: A next step: Visualizing errors and uncertainty. *Computer Graphics and Applications, IEEE 23*, 5 (2003), 6–10. 3

[Loh10] LOHR S.: *Sampling: Design and Analysis*, 2nd ed. Brooks/Cole, Boston, MA, 2010. 2, 3, 7

[MFMe10] MORELAND K., FABIAN N., MARION P., ECI B. G.: *Visualization on Supercomputing Platform Level II ASC Miles tone (3537-1B) Results from Sandia*. Tech. Rep. SAND2010-6118, Sandia National Laboratory, Sept. 2010. 2

[pan] Panasas resource library. http://www.panasas.com/products/library.php#productinfo. 2, 8

[SBH*10] SZALAY A. S., BELL G. C., HUANG H. H., TERZIS A., WHITE A.: Low-Power Amdahl-Balanced blades for data intensive computing. *ACM SIGOPS Operating Systems Review 44*, 1 (2010), 71. 2

[TCM10] TIKHONOVA A., CORREA C., MA K.: Visualization by proxy: A novel framework for deferred interact ion with volume data. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (2010), 1551–1559. 2

[TYR*06] TU T., YU H., RAMIREZ-GUZMAN L., ACOBO BIELAK J., GHATTAS O., MA K., O'HALLARON D. R.: From mesh generation to scientific visualization: An End-to-En d approach to parallel supercomputing. p. 91. 2

[WHA*10] WOODRING J., HEITMANN K., AHRENS J., FASEL P., HSU C. H., HABIB S., POPE A.: Analyzing and visualizing cosmological simulations with ParaVi ew. *Arxiv preprint arXiv:1010.6128* (2010). 7