ETH: A Framework for the Design-Space Exploration of Extreme-Scale Scientific Visualization

Gregory D. Abram
Texas Advanced Computing Center
gda@tacc.utexas.edu

Vignesh Adhinarayanan
Virginia Tech
avignesh@vt.edu

Wu-chun Feng
Virginia Tech
feng@cs.vt.edu

David H. Rogers
Los Alamos National Lab
dhr@lanl.gov

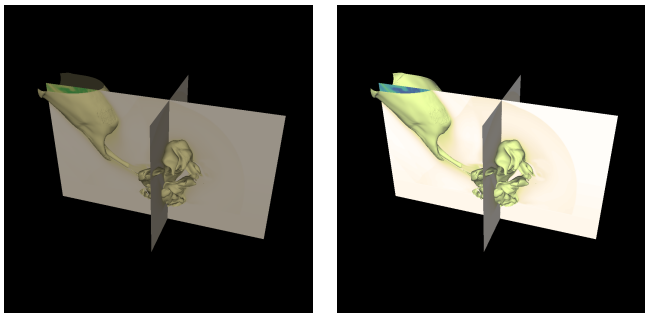
James P. Ahrens
Los Alamos National Lab
ahrens@tacc.utexas.edu

Luke A. Wilson
Texas Advanced Computing Center
lwilson@tacc.utexas.edu

ABSTRACT

As high-performance computing (HPC) moves towards the exascale era, large-scale scientific simulations are generating enormous datasets. A variety of techniques (e.g., in-situ methods, data sampling, and compression) have been proposed to help visualize these large datasets under various constraints such as storage, power, and energy. However, evaluating these techniques and understanding the various trade-offs (e.g., performance, efficiency, quality) remains a challenging task.

To enable the investigation and optimization across such trade-offs, we propose a toolkit for the early-stage exploration of visualization and rendering approaches, job layout, and visualization pipelines. Our framework covers a broader parameter space than existing visualization applications such as ParaView and VisIt. It also promotes the study of simulation-visualization coupling strategies through a data-centric approach, rather than requiring the code itself. Furthermore, with experimentation on an extensively instrumented supercomputer, we study more metrics of interest than was previously possible. Overall, our framework will help to answer important *what-if* scenarios and *trade-off* questions in early stages of pipeline development, helping scientists to make informed choices about how to best couple a simulation code with visualization at extreme scale.



(a) Raycast image (b) Geometry-based image

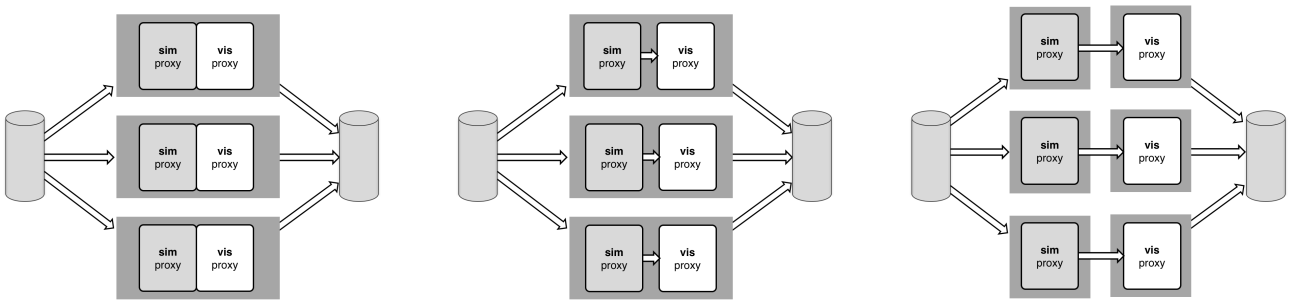
Figure 1: Sample images from datasets tested and rendered with the ETH toolkit. Different rendering pipelines, coupling strategies, and sampling approaches can all be tested in various configurations with the toolkit. Here we show both a raytraced and geometry-based rendering of an asteroid impact from an xRAGE simulation.

1 INTRODUCTION

Power, storage, and data movement have emerged as first-order design constraints in supercomputing systems. For instance, the U.S. Department of Energy has imposed a power budget of 20-30 MW for the candidate exascale systems [1]. The applications running on these machines are also limited by their aggregate I/O bandwidth (60 TB/s) and storage capacity (1000 PB) [1]. Such constraints hamper our ability to visualize and analyze extreme-scale datasets via conventional methods [25, 26]. In response, researchers have developed numerous techniques (e.g., in-situ methods [3], data sampling [35], compression [20], and job layout optimization [4, 29]) to address the challenges associated with large datasets. Our goal in this paper is to help domain scientists choose the best approach for a specific challenge in a complex space of design trade-offs.

Scientists using *traditional* workflows have developed rules of thumb for how to sample, store, and visualize their data. However, such rules will *not* apply in the exascale era due to the sheer number of options available for in-situ methods [3]. In addition to visualization and rendering methods, emerging techniques for managing power [19] and data movement [10] will create an even more complicated and bewildering array of options. The complexity of the workflows that are possible from these options will be enormous. Conventional evaluation methods using a fully-featured production software (e.g., Paraview [8] or VisIt [18]) and tight coupling between the simulation and visualization codes will restrict rigorous experimentation to only a small number of workflows. A poor understanding of the trade-off space may result in data and visualization products that are not useful for specific tasks. For instance, some methods of sampling or compressing data may remove features of data or impact the perception of those features such that the data becomes useless. Therefore, to facilitate the exploration of a broader design space, we present a flexible toolkit called the *exploration test harness* (ETH).

Using ETH, one can conduct experiments on real scientific data, real analysis and visualization operations, and real rendering approaches with an implementation that promotes computation on real data, but decouples the testing from applications. This promotes flexibility while maintaining a tie to useful analysis and visualization operations. The approach is based on insights gained from our earlier experiments that have shown when studying in-situ approaches, the behavior of the application can be factored out [32]. So one can run successful experiments without directly coupling the code with the analysis and visualization operations that one would like to study. The critical component of the pipeline is the *data*, rather than the coupling with applications. While it is true that coupling with applications can provide some additional information, the trade-off space explored through ETH is complex enough on its own and the results valuable enough to merit the simplification of decoupling from the application. The specific contributions of our *experimental*



(a) Tightly coupled sim/vis proxies (b) Loosely coupled sim/vis proxies (cores). (c) Loosely coupled sim/vis proxies (nodes).

Figure 2: Diagram showing the three types of coupling available through ETH. The general unit of operation is a *simulation proxy*/*visualization proxy* pairing, and many of these are run at once to achieve a parallel test. Tests can be run on (a) tightly coupled pairings, in which the *simulation proxy* and the *visualization proxy* are in the same process, (b) loosely coupled inter-core pairings, in which the *simulation proxy* and the *visualization proxy* are on the same node, but different cores, and (c) loosely coupled inter-node pairings, in which the *simulation proxy* and *visualization proxy* are on different nodes. In all cases, data is read from disk into *simulation proxies*, where optional analysis and visualization operations can occur before the data is sent to *visualization proxies*, where more operations can occur before the data are rendered to output artifacts.

test harness (ETH) are as follows:

- **A first-of-its-kind toolkit for answering "what if" questions**. This framework will facilitate a deep understanding about the trade-offs among different operations, sampling, visualization pipelines and coupling.
- **A toolkit that runs on data rather than the coupling to a science code**. While other toolkits can also be used to operate directly on the data, the design of ETH ensures that we can even answer questions pertaining to coupling optimizations without real sim-viz code coupling (see Fig. 2). This is an important aspect to flexibility, as it means experiments can be run without investment in coupling the in-situ pipeline with a specific code.
- **Built-in exploration of multiple rendering approaches**. In particular, the toolkit includes a raycasting approach that operates on the raw data and a geometry-based approach that performs traditional triangle-based operations (see Fig. 1). The raycasting approach holds great promise for data at scale, and current work by vendors makes this a viable alternative.
- **Results from experiments on representative data types (grid and points) across different configurations**. In particular, we have conducted experiments on two classes of data — points and grids — that represent the types of data that extreme-scale applications create. Our experiences with both traditional evaluation and ETH showed that design space exploration with ETH was an order of magnitude faster due to the simplified process.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents an overview of our ETH framework and section 4 describes the parameter space explored in this paper. The experimental setup and results are presented in sections 5 and 6, respectively. Section 7 concludes our paper.

2 Related Work

Scientific visualization techniques have been used for a long time and therefore, a number of visualization frameworks for in-situ and post-processing visualization exist. Early instances of in-situ implementations focused on tightly integrating a visualization routine with a simulation routine (e.g., pV3 [14]). These implementations are typically customized for a specific purpose such as for monitoring or steering a simulation. Some modern frameworks, on the other hand, provide a great deal of customizing ability. Despite their customization capabilities, contemporary visualization frameworks are largely ill-suited for rapid design-space exploration. A majority of the visualization frameworks can be classified into one of the following two categories:

- **They are designed for a specific purpose and narrow in scope.** `CUMULVS` [17], `EPSN` [6], and `SCIRun` [15] are task-specific for computational steering, `yt` [31] is data-type specific for AMR data, `Nessie` framework [23] is designed to be application-specific, and `QIso` [36] is for extracting and operating upon surfaces. Owing to their narrow focus, they are ill-suited for design-space exploration.
- **Their focus is on integration with a real-world application rather than exploratory research.** Some examples in this category include `Catalyst` [2], `ADIOS` [24], `GLEAN` [33], `VisIt` [18], and `Damaris/Viz` [5], and `Libsim` [37]. While these frameworks provide a vast array of options to explore the in-situ visualization space, the effort required to use these tools for design-space exploration is significantly high. For instance, to get an in-situ setup running with `Catalyst` [9], one has to compile and configure `Catalyst`, the simulation code, and write a custom adapter. This added complexity makes this class of in-situ framework ill-suited for exploration of the design space.

Some of the other frameworks are designed to encourage exploration by easing the burden of integration through simplified design or by narrowing the tool's focus (e.g., by operating only with mini-apps). `Strawman` [21] belongs to this category with one of its stated goal being to explore in-situ solutions. Much like Catalyst, it also requires integration with applications (or mini-apps) by writing adaptors. Like the full-fledged visualization toolkits, using such tools for design-space exploration would also be time-consuming due to the additional time cost involved in writing adaptors, compiling and running the simulation. Our ETH framework simplifies the evaluation process by eliminating the need for writing adaptors, compiling and running the simulation, and coupling data management frameworks

(e.g., ADIOS and Damaris) with visualization frameworks (e.g., ParaView/Catalyst and VisIt).

Why not use existing toolkits in the "post-processing" mode and directly operate on data? In our earlier design-space experiments, we attempted to use ParaView/Catalyst to perform the study directly on the data. However, our experiences with the setups presented in Fig. 2 showed that these setups are not appropriate for ParaView/Catalyst which lacks good transport mechanisms for the data. One would have to setup Catalyst with a data management framework like ADIOS or write one's own transport code, neither of which are simple tasks [11]. Furthermore, getting custom rendering working with existing frameworks is a more involved process compared to a tool such as ETH.

3 Exploration Test Harness (ETH)

The Exploration Test Harness (ETH) is a lightweight, open-source testing harness that promotes exploration of a variety of analysis and visualization pipelines in many different operations, work distributions, and mappings onto hardware. The toolkit is based on VTK [30], which is a core capability for the analysis and visualization community. VTK implements a data-centric pipeline of operators, filters and rendering operations that operate on data, then pass it along to the next element in the pipeline. There are several important capabilities of the test harness, which are discussed below. These capabilities are the classes of parameters that are varied in the tests run for this paper.

ETH operates on data and does not require code coupling. ETH loads test data from disk, and this has several advantages over other systems. First, we can run the framework on many different science domains, without changing the framework. Because the toolkit is based on VTK, any dataset that VTK reads can be used in tests. Existing toolkits tend to be based on *proxy apps*, which contain simplified versions of the physics of their related simulation. Operating on real data is critical to testing the visualization and analysis operators as simulated data does not generally contain enough complexity to test and exercise interesting analysis and visualization operations.

ETH has easily configurable visualization operations. A critical feature of a useful testing system is a way to modify analysis and visualization operations so that they approximate those that are to be tested. Again, since ETH is based on VTK, many operations can be easily added to the pipelines tested, and they can be specific to the data and visualizations that are of interest. This means that ETH can generate specific visualization products that serve as good proxies for perceptual and cognitive test products.

ETH can run with different process-couplings. Given the number of different hardware and process options available today, it is useful to study different configurations of parallel jobs. As shown in Figure 2, ETH can be configured to run pairs of *simulation proxy* and *visualization proxy* processes in three ways: 1) coupled into a single process, or 2) as communicating processes, which enables the processes to run on the same nodes, thereby avoiding any I/O operations, or 3) on different nodes, enabling testing using differing numbers of nodes and on heterogeneous systems.

ETH can run different rendering pipelines. Traditional visualization workflows use algorithms that iterate over the input data to extract intermediate geometrical representations of the data that can then be rendered using OpenGL, which then iterates over the intermediate data to determine each element's contribution to the output image. Recent technical advances [16, 27, 34] make it practical to support raycasting renderers that operate directly on data, avoiding the need for intermediate representations and the memory space they require. Further, in many cases, raycasting can produce significantly better images at lower cost, particularly as datasets get large. Since the choice of visualization back-ends significantly affects a system's overall performance and the effectiveness of the results, ETH can be configured for alternative rendering back-ends as shown in Figure 4, enabling this dimension to be explored as it affects the overall system performance.

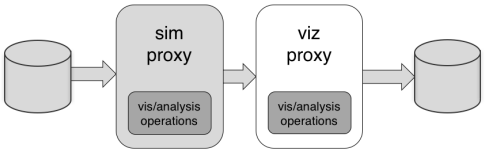


Figure 3: Diagram showing the basic unit of ETH, a *simulation proxy/visualization proxy* pairing. Data is loaded from the disk, sent to the *simulation proxy*, and then passed from there to the *visualization proxy*. From there it is written to disk. Visualization and analysis operations can be performed in either place, depending upon the pipeline that is being tested. This means that ETH can compare sets of operations weighted towards either.

3.1 Design of the Framework

ETH is a parallel execution framework, but the basic unit of the toolkit is a pair of processes, each of which can perform analysis or visualization operations on real simulation data as shown in Figure 3. The toolkit reads data from the disk and then operates on the data in parallel. An experiment can then be run over different process or node configurations, and different rendering pipelines to produce artifact on disk. The system can accommodate a wide range of analysis and visualization operations in either a simulation process or an analysis process, so the toolkit can be customized to operate on a users specific data, and perform visualizations that are simplified or specifically tuned to the user's work flow.

Modes include unified (single process), core-to-core, and node-to-node. There are two rendering pipelines supported as well raycasting (non-geometric) and geometry-based rendering.

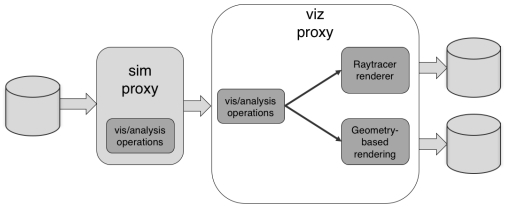


Figure 4: Diagram showing options for pipeline execution inside the vis process. We can send the result of some set of operations to different rendering pipelines. In this case, we can compare a traditional geometry-based rendering approach with a raycasting approach that operates directly on the data, and takes advantage of new software rendering libraries that are optimized for this.

3.2 Execution Details

As described above, ETH runs with pairs of simulation and visualization proxies coupled together into one process, or running in separate

processes and communicating via the socket layer. In the first case, experiments are easily run using the standard batch scheduler.

When the simulation and visualization proxies run in *separate* processes, the two sets of proxy processes must exchange information that enables the visualization proxy processes to connect to their paired simulation proxy processes. This is done by starting the parallel application in two steps: first, the simulation proxy application is started. Each process of the application then adds its assigned IP address and port number to a globally accessible layout file, then opens its port and waits for connection. The visualization proxy application is then started. Each process of the application then references the global layout file, determines the location of the simulation proxy(s) it will receive data from, waits for the corresponding port to open, and then establishes the connection.

When the simulation and visualization proxy applications run on the same nodes, this is again easy to start: the job script simply begins the simulation proxy application first, in the background, and then starts the visualization proxy application in the foreground.

When the simulation and visualization proxy applications run on separate sets of nodes, several alternatives exist depending on the mix of nodes required; on homogeneous systems, a single job allocating the total number of nodes required, and MPI arguments can be used to start the two parallel processes offset from one another. When heterogeneous collections of nodes are desired, it will be up to the scheduling system to arrange for two separate jobs to be started concurrently.

4 METHODOLOGY

There are several configurable parameters that affect the performance, power, energy, and quality of the images produced by the visualization process. In this paper, we study three such parameters that we believe will lead to insightful results. The rest of this section describes these parameters and the applications used in this study.

4.1 Applications and Input Datasets

In this paper, we explored three different datasets. A point-based dataset from a cosmology simulation, a grid-based dataset from an asteroid impact simulation, and a testing dataset. These are representative of large classes of science datasets of interest at extreme scale. As part of this test, we prepared three different input datasets for each domain, to approximate sampling done by the simulation before it is handed to the visualization operators. The applications and datasets are described in detail next.

Cosmology Simulation (HACC) Hardware/Hybrid Accelerated Cosmology Code (HACC) simulation is a cosmological n-body simulation used to study the evolution of the universe [13]. This simulation helps in understanding the distribution of dark matter within a halo which in turn plays a major role in interpreting the results from real-world dark-matter observation studies typically performed with high-resolution telescopes. As such, visualization of halos forms an important step in this scientific workflow.

Particle data for the billions (or trillions) of particles from the n-body simulation is passed on as input to the in-situ visualization engine in each time-step. Each particle's data is composed of its id, position vector, and velocity vector. The visualization task here is to render the point-cloud data in a manner that makes visual identification of halos easy. Many different techniques can be used to perform this simulation, each with different performance, power and energy consumption, and perception characteristics (i.e., image quality). A sample visualization of the halos rendered using the raycasting method is presented in Figure 5b. For our experiments, we use four datasets from the dark sky simulation, the largest of which represents over a billion particles and the smallest comprising nearly a quarter billion particles.

Asteroid Simulation (xRAGE) Radiation Adaptive Grid Eulerian (xRAGE) is a radiation-hydrodynamic code used for solving a variety of high-deformation flow problems involving a radiative transfer of energy (e.g., underwater blast) [12]. This application is employed in many domains such as astrophysics, where the impact of asteroid collision on the ocean is studied [12]. The simulation proceeds by solving a series of Euler equations and radiation equations using mesh-based computation techniques. While the simulation itself normally uses adaptive mesh refinement (AMR) method, the AMR data is typically converted to an unstructured grid data which is then downsampled to a structured grid data before being handed off to the visualization code. The visualization task is to represent a grid of variables such as pressure, density and (in our case) temperature. A sample image from an asteroid collision simulation is presented in Figure 5a. Such visualizations are useful in verifying the correctness of the simulation and to quickly understand the difference in results across various runs of an ensemble of simulations with different initial conditions. While a variety of techniques exist to visualize a uniform grid, identifying the best possible rendering in an increasing cost-constrained HPC environment becomes necessary. Therefore, we analyze this type of data with our ETH design-space exploration tool. The data sizes used in this study range from 0.2GB to 8GB per timestep.

Configurable Test Data In addition to studying real-world datasets of practical relevance, our framework also allows the user to perform controlled studies by configuring test datasets. We present results for one such case where we generated dataset using a time-varying 3D Perlin coherent noise generator [28]. This enables us to control the frequency content and distribution of the data. This data was generated at a resolution of 1024^4 512^4 and 256^4.

4.2 In-situ Parameters

Once the data have been read into the *simulation proxy*, We study two in-situ methods, namely, sampling and coupling strategy. The various options for each parameter are described next.

Sampling Technique Spatial sampling is explored which operates by selecting a subset of points (down sampling) from the original dataset based on some given distribution. We vary the sampling ratio (i.e., number of selected points from the given "raw" data) and study how the various metrics included in this study changes.

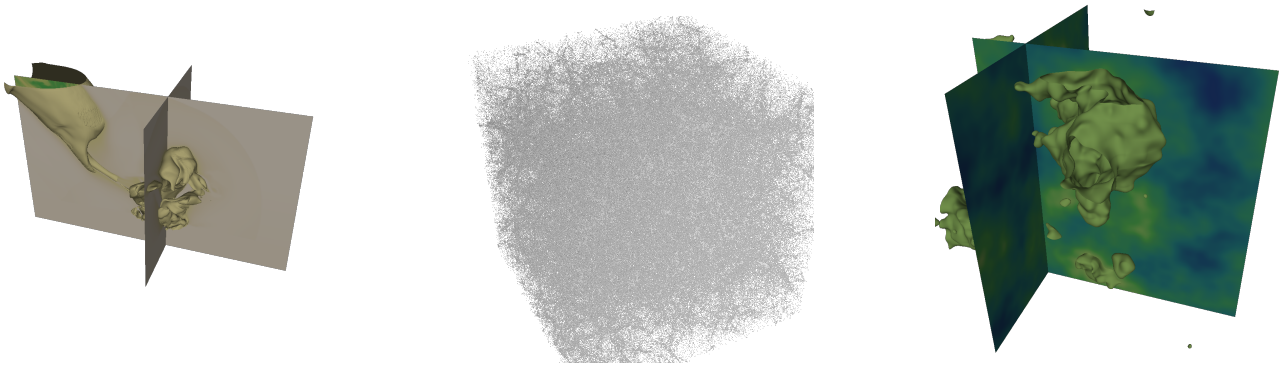
Coupling strategy We explore three work-distribution or sim-viz coupling strategies.

- *intercore* coupling – Simulation and visualization processes are time-shared and alternate on same set of nodes.
- *internode* coupling – The processes are space-shared with the simulation process running on half the number of allocated nodes and the visualization process running on the remaining nodes
- *tight* coupling – the visualization and simulation processes are merged to create a single, unified process for running a scientific workflow

These coupling strategies are expected to have distinct advantages based on the scalability of the rendering technique and data size. We seek to quantify the impact of the coupling strategy on our selected metrics.

4.3 Rendering Parameters

Once the data have passed through the operations requested by the test designer, they are rendered to disk. There are two pipelines available in ETH: (1) a geometry-based renderer, and (2) a raycasting renderer. The geometry-based rendering pipeline utilizes VTK to generate geometry from the data, and then render that geometry in a



(a) Asteroid sim. (xRAGE) (b) Cosmology sim. (HACC) (c) ETH Test Data

Figure 5: Sample images from the datasets tested with this frameworks. An asteroid impact (a) was created with xRAGE, and a cosmology simulation (b) was created with HACC, and (c) was created in ETH, using a Perlin noise function.

variety of ways. The raycasting method operates directly on the data rays from a light source (e.g., camera) to all the pixels in the image plane placed in front of the object. If the ray passing through the pixel hits an object, the color for that pixel is calculated and rendered. If the input object is a point cloud, the data is preprocessed into an appropriate data structure so that it becomes possible for the rays to hit an "object." Note that unlike the other two methods, this method is dependent on the number of rays cast, once the initial data structure is built.

Rendering Methods for HACC data The HACC data consists of a very large number of particles; essentially, points in 3D space. Ideally, these points will be rendered as spheres, so that distant particles will be smaller to give the appearance of distance. This case presents a particularly difficult case for traditional geometry-based visualization, since modeling each of a billion or more spheres using sufficient triangles to give the appearance of roundness is impractical. Instead, we use two alternative rendering techniques to test geometry-based rendering.

Geometry-based: VTK Points. This is the simplest of the techniques used in this paper. The technique operates on a collection of points (also referred to as a *point cloud*). Each point in the collection is described by its position in the *x*-*y*-*z* plane. Usually, the location of the point in the 3-d space is also accompanied by a scalar field. The technique operates by mapping every point in the 3-d space to the 2D plane and rendering every pixel with a fixed size (usually 1 to 3 pixels on a side) and fixed color block. This normally results in a loss in 3-D perception.

Geometry-based: Gaussian Splatter. This technique creates a single triangle, sized to reflect a given radius, to represent each point. At rendering time the triangle is transformed to the proper location in eye-space, oriented toward the viewer, and rendered to the screen using a specialized shader function that manipulates the triangle normal at each pixel to model a sphere. In its current implementation this leads to some unfortunate artifacts, but it does limit the amount of geometry required to represent the data for rendering.

Raycast Spheres. This case is particularly well-suited to raycasting. Each particle is represented as a 3D point and a world-space radius, and placed into an specialized acceleration structure at a cost of roughly O(NlogN). At run-time, the acceleration structure is traversed to determine whether the viewing rays (that is, rays begun at the viewpoint passing through each pixel in the image plane) strikes a sphere with a cost that is sub-linear in the number of particles. If a ray *does* intersect a sphere, a simple geometric calculation produces an intersection depth and orientation for shading.

Rendering Methods for Volumetric Data. Both the asteroid and synthetic datasets are scalar and volumetric; in the case of the asteroid data, the data represents the temperature field in the vicinity of the steroid strike, while the synthetic dataset represents coherent (Perlin) noise in an arbitrary three dimensional space. In our tests we use two traditional visualization techniques that are very widely used in such cases: slicing planes and isosurfaces.

Slices and Isosurfaces in Geometry-based Visualization. To visualize slices and isosurfaces of volumetric data, geometry-based visualization systems must first generate geometry representing the slice or isosurface as a set of triangles, which are then rendered using a standard OpenGL pipeline. Generating such geometry consists of two steps: first, identifying the cells of the data grid that contain fragments of the surface, and then determining the geometry within those cells. While efficient algorithms attempt to minimize the number of cells that intersect the surfaces, a large number of the input cells will be examined and and a very large amount of geometry will often be created. In the case of slicing planes, the work and resulting data size is proportional (roughly) to the 2/3 root of the input data size, while in the case of isosurfaces the work is proportionate to the data size and the size out geometry set can range from zero to proportionate to the input data size.

Slices and Isosurfaces in Raycasting. Again, raycasting provides an attractive alternative to geometry-based visualization in these cases. The intersection of an arbitrary ray with an implicitly defined plane to produce a hit point in data space is O(1), and (in the case of structured grids) looking up the corresponding data value is also O(1), so the cost of rendering slicing planes is O(number of pixels). Isosurfaces are rendered by iterating along each view ray, sampling to find the data value for each iteration, and looking for crossings. Once a crossing is found, a hit point can be interpolated. Note that the appropriate sampling along the ray is proportionate to the resolution of the data in 1D, so the cost of each ray is proportionate to the 1/3 root of the input data size.

5 Experimental Setup

We run our experiments on *Hikari*, a 432-node HPE Apollo 8000 cluster, which is capable of delivering over 400 TFLOPs of peak performance. Each node of the cluster is equipped with *two* 12-core Intel Haswell E5-2600v3 processors operating at 3.5 GHz and up to 64GB of RAM per node. The nodes are interconnected by Mellanox EDR infiniband using a fat tree topology. The cluster operates on high-voltage direct current (HVDC) instead of the traditional alternate current thereby avoiding four AC/DC conversions. This results in the cluster consuming significantly lesser power at both idle

and active modes than a similar-sized cluster operating on alternate current. The system is extensively metered to allow quantitative studies. Apollo 8000 system manager is used to record records power every 5 seconds on a half-rack basis. We use *TACC stats* [7], a low-overhead monitoring infrastructure, to collect hardware performance counter data, which we use for analyzing our results.

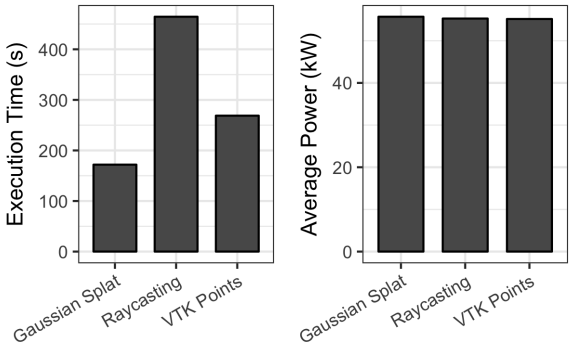
The software setup is as follows: we use CentOS Linux 7.2.1511 running kernel version 3.10.0-327.13.1 IMPI 5.1.2 is used to parallelize our jobs across sockets, TBB 4.4.1 is used for threading, Intel ISPC is used for vectorization.

6 Experimental Results

In this section, we present experimental data for the three datasets previously described. The primary goal of this section is to demonstrate the usefulness of our design-space exploration framework rather than perform an in-depth characterization study.

6.1 Cosmology Simulation (HACC)

Performance, power, energy, and relative accuracy of visualization artifacts is presented here.



(a) Performance (b) Average Power

Figure 6: Execution time and average power consumption of raycasting, Gaussian splat, and VTK points algorithms for the cosmology application

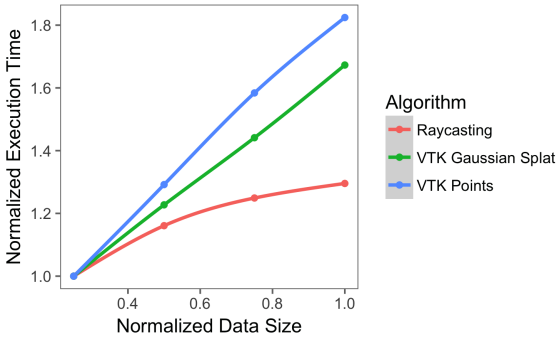


Figure 7: Scalability of raycasting, gaussian splat, and VTK points with respect to problem size. Results are normalized to the smallest problem size for each algorithm.

Algorithms. Figure 6 presents the execution time and average power consumption for the *raycasting*, *Gaussian splat*, and *VTK points* for the *large* dataset in our test suite. As shown in Figure 6a, the execution time for the Gaussian splat algorithm is 36% lower than VTK points which itself is lower than raycasting by 42%. An analysis of performance counter data shows that the raycasting algorithm performs significantly more computations than the other two algorithms for the problem size considered. Most of the additional computations come from an additional setup phase where an acceleration structure is built. Gaussian splat executing faster than VTK points is an unintuitive result as they both render all the points presented to them as input, but Gaussian incurs an additional step where the points are splatted to nearby voxels. Thus, in theory, this algorithm performs more computations than VTK points. We attribute its fastness to a superior implementation of Gaussian splat. Note that while the raycasting algorithm is the slowest, the quality of the images rendered by this algorithm tends to be better than the other two algorithms. Quantifying the perceptive value of the image produced, however, is an active research problem beyond the scope of this paper.

Observation 1. Gaussian splat is faster than VTK points which in turn is faster than raycasting.

The power consumption remains relatively constant across the three algorithms with only Gaussian splat exhibiting a marginally higher consumption as shown in Figure. 6b. However, this marginal difference is insignificant and should be attributed to noise in the measurement. Similar power consumption for the algorithms is understandable as we expect the resources to be utilized to the same levels. Raycasting is fully parallelized – MPI is used to parallelize across the nodes, Intel TBB is used to parallelize across the cores within a node, and Intel ISPC compiler is used to vectorize across SIMD lanes; same is the case with VTK library implementations. As long as the problem size remains large enough to occupy all the resources, the implementations will utilize the hardware to similar levels and consume the same power. With power being effectively the same across the three algorithms, the energy consumption tracks performance (not explicitly shown in Figure 6).

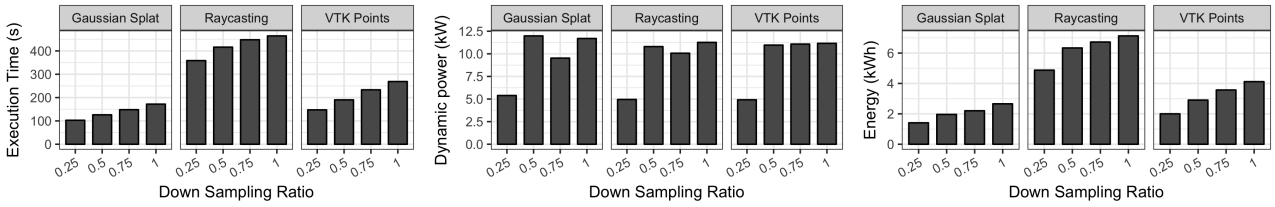
Observation 2. Power consumption remains nearly the same for raycasting, Gaussian splat, and VTK points.

Scalability of the algorithms with data size. We present the *normalized* execution time for the three algorithms as we change the data size and fix the number of nodes in Figure 7. The normalization is performed against the smallest dataset within each algorithm. Gaussian splat and VTK points show a linear increase in execution time as the problem size increases as they both run in O(n). Of these two algorithms, VTK points scale better than Gaussian splat. However, for all the problem sizes considered in this study, Gaussian splat continued to outperform VTK points. Raycasting, unlike Gaussian splat and VTK points, shows a sub-linear increase in execution time as the performance of raycasting is not dependent on the number of points, but the number of rays. While the initial structure-generation phase of raycasting is affected by the number of points, all the other stages depend on the number of rays cast which remains constant. Therefore, we expect raycasting to outperform the other two algorithms for problem sizes larger than those considered in this paper.

Observation 3. Geometry-based methods exhibit a significantly different scaling curve when compared to geometry-free method which indicates that the optimal algorithm is dependent on the problem size or cluster size.

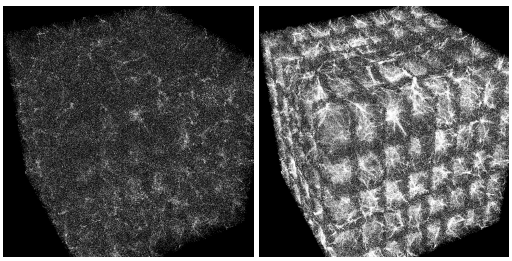
This finding is also corroborated by Larsen et al. where they report that raycasting outperforms rasterization for larger problem sizes but not for smaller problems [22]. Our framework makes such discoveries easier as it removes the necessity to integrate visualization frameworks with simulation codes.

Sampling Techniques. Figure 14 presents the performance, dynamic power, and energy consumption for four different spatial-



(a) Performance (b) Power (c) Energy

Figure 8: Performance, power, and energy consumption for four different spatial sampling configurations for the cosmology application



(a) Sampling Ratio: 0.25 (b) Sampling Ratio: 1.0

Figure 9: HACC images at two different sampling ratios. Gaussian splat technique was used to generate these images.

sampling ratios for the three different algorithms. Spatial sampling ensures that fewer points are rendered, so the execution time goes down as the sampling ratio increases as shown in Figure 8a. The result worth noting here is the dynamic power consumption for the different sampling ratios shown in Figure 8b. At low sampling rates (less than 0.5), we see a significant reduction in power consumption. More specifically, the total power consumption at a sampling rate of 0.25 is 11% lower than the power consumption for the full dataset (i.e., sampling ratio = 1.0). This value actually corresponds to a 39% reduction in the dynamic power of the system. Our hypothesis is that at this point we have reduced the problem size to a level where it becomes difficult to keep all of the parallel resources busy thus affecting the resource utilization.

Observation 4. Spatial sampling is a viable method to reduce system power consumption at the cost of some loss is perception.

A side effect of reducing sampling ratio is a reduction in the quality of image which may reduce its perception value. A visual comparison between a full image and a sampled image is shown in Figure 9. We also quantify the error introduced by the sampling technique using the root mean square error (RMSE) metric in Table 1. While the energy saved increases with loss in accuracy, as expected, the trade-off curve between the two differ across algorithms significantly. For example, VTK points show the most resilience to error when spatial sampling is applied to a dataset. It's RMSE is as low as 0.13 when one in four points are sampled, whereas for the other algorithms, it is at least 0.42. However, the baseline image (i.e., no sampling is applied) for VTK points is generally worse than the other two. We can also observe from the table that raycasting shows more tolerance to errors at higher sampling rates, but the errors creep up as we sample less. While we have used a simple metric, in practice, we expect users of the framework to use more sophisticated metrics explicitly targeted at measuring the perception quality of an image.

Strong Scaling Results. Figure 10 shows the results for two different node counts. For brevity, we discuss only the "full" dataset. The other datasets within the cosmology application follow similar trends. In this figure, we observe that the performance of the raycasting algorithm improves only slightly when we increase the node count as shown in Figure 10a. The average power consumption when 200 nodes are used is nearly 50% lower than when 400 nodes are used resulting in a similar magnitude of energy saved as shown in Figures 10b and 10c. This is true for all three algorithms studied.

Table 1: Trade-off between accuracy and energy for HACC

Algorithm	Sampling Ratio	0.75	0.50	0.25
Raycasting	RMSE	0.17	0.28	0.42
	Energy Saved	17.4%	28.1%	41.5%
Gaussian Splat	RMSE	0.33	0.37	0.43
	Energy Saved	17.2%	26.3%	47.0%
VTK Points	RMSE	0.04	0.08	0.13
	Energy Saved	13.3%	29.4%	51.4%

Observation 5. For the problem sizes considered, visualization algorithms such as raycasting, Gaussian splatter, and VTK points exhibit poor strong-scaling.

An implication of the above observation is that the commonly used coupling strategy, where simulation and visualization alternates on all the nodes of a supercomputer, may not be the most energy-efficient, as the additional nodes largely "waste" power and energy. We posit that a better way to distribute work is be to allocate a small number of nodes for visualization and the remaining nodes for simulation and space share the nodes between the two.

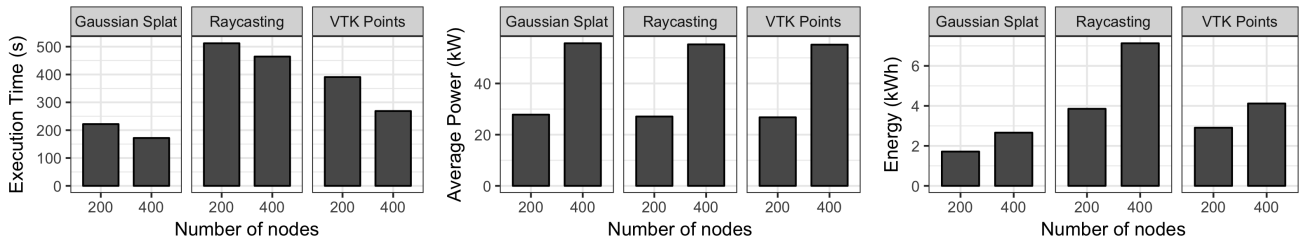
Coupling Strategies. We verify our hypothesis by performing an experiment with different coupling strategies. As shown in Figure 11, tight- coupling strategy (tightly coupled and intercore coupled) is not always optimal both in terms of performance and energy consumption. This set of results belies expectations and underscores the need for experiment- and data-driven decision making. Our software framework along with the experimental hardware platform makes experimentation and data collection easier and facilitates rapid design-space exploration.

Observation 6. Proximity between the simulation and visualization routines does not determine optimality as evidenced by the intercore coupling which outperforms the other coupling strategies for the HACC application.

6.2 Asteroid Simulation (xRAGE)

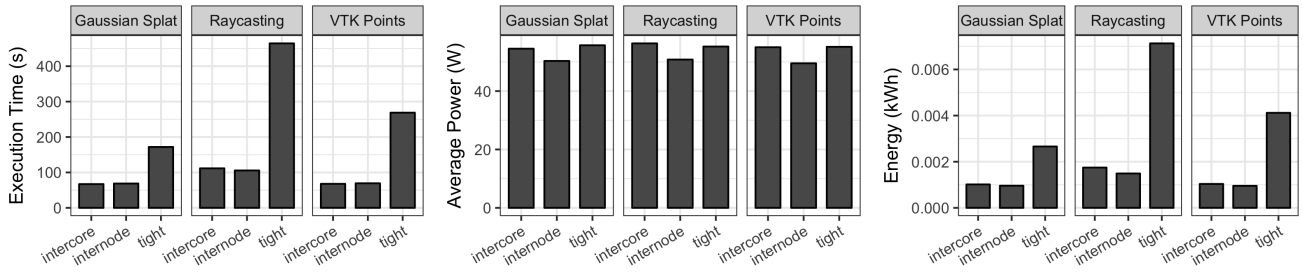
Experimental results are discussed for the xRAGE application in this section. Since this application operates on a structured grid instead of HACC's point-based data, the algorithms considered for evaluation differs.

Algorithms. Figure 12 shows the performance, power, and energy consumption for VTK's geometry-based isosurface algorithm (vtk)



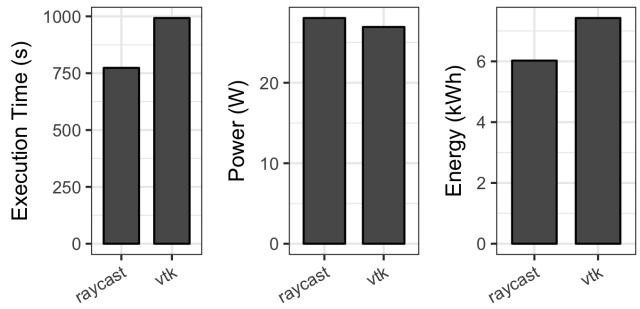
(a) Performance (b) Power (c) Energy

Figure 10: Experimental results for two different node counts for the cosmology application



(a) Performance (b) Power (c) Energy

Figure 11: Experimental results for different simulation-visualization coupling strategies for the cosmology application



(a) Performance (b) Average Power (c) Energy

Figure 12: Execution time and average power consumption of raycasting and vtk algorithm for the xRAGE application

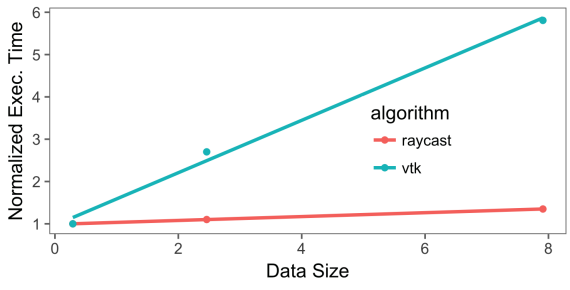


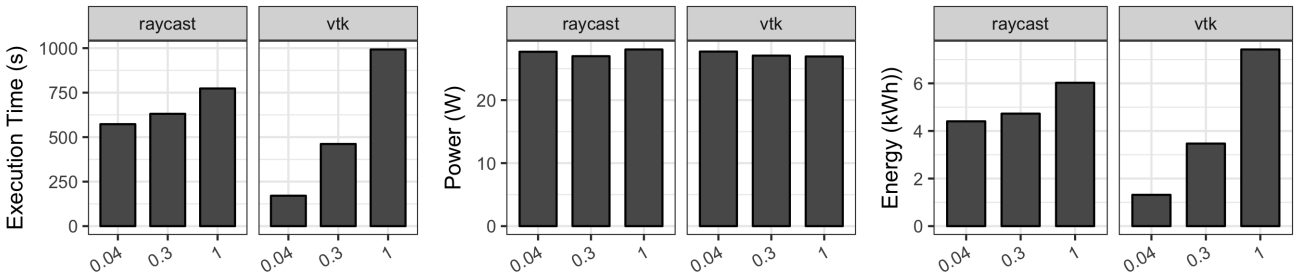
Figure 13: Scalability of raycasting and geometry-based (vtk) methods with respect to problem size. Results are normalized to the smallest problem size for each algorithm.

and raycasting. As we can see from this figure, vtk takes 28% more time than raycasting to generate the isosurface. While VTK's implementation consumes lesser power than raycasting (Figure 12b), it is offset by a significant increase in execution time resulting in higher energy consumption for VTK as shown in Figure 12c.

Scalability of algorithms with problem size. Though both VTK's and raycasting's execution time increases in O(Data Size), the slope of the line is significantly different. A 27-fold increase in problem size resulted in VTK taking 5.8 times longer to execute, whereas for raycasting it was only a 1.35-fold increase. In fact, VTK executed faster for the smallest problem size, but the trend reversed when the data size was increased by a factor of 2 in all three dimensions of the 3D grid.

Sampling Technique. We previously identified spatial sampling as a technique to reduce power consumption. Here we run the sampling experiments with the asteroid dataset. As observed in Figure 8b, power consumption does not reduce with sampling ratio even when the sampling ratio is reduced to 0.04 (compared to up to 0.25 in the cosmology application). This indicates that optimization techniques (for both power and performance) are not common across domains. While sampling helped reduce power for HACC, it only helps in reducing energy for xRAGE (shown in Figure 12c.

Strong Scaling Results. Figure 15 shows an interesting trend. In this graph, we plot the normalized performance (proportional to the reciprocal of execution time) versus the number of nodes used in the experiments. We varied the node count from 1 to 216. We can see that the raycasting algorithm scales well. When we double the number of nodes, the performance roughly doubles, which is its expected property. VTK on the other hand, does not only fail to scale, but actually shows performance degradation beyond a point. We think this is due to some form of contention in a shared resource arising from parallelism.



(a) Performance (b) Power (c) Energy

Figure 14: Performance, power, and energy consumption for three different spatial sampling configurations for the xRAGE application

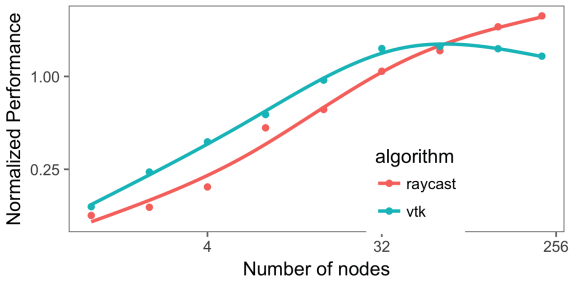


Figure 15: Strong scaling curve for raycasting and VTK geometric rendering for the xRAGE application.

6.3 Configurable Test Data

The configurable test data is very similar to the cosmology application. The observations made for the application largely applies here as well due to the similar datasets and the algorithms being explored. Therefore, we present one example results only that shows the effect of sampling on the overall performance (i.e., execution time) in Figure 16. We can also observe that the gap in performance between raycasting and vtk rendering diminishes as the problem size increases. This corroborates with all previous experiments.

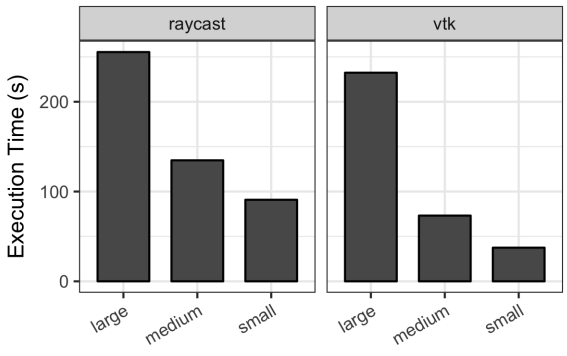


Figure 16: Execution Time of the Test Dataset

7 Conclusion and Future Work

In-situ methods are becoming an indispensable part of the scientific workflow at extreme scale and the number of options for these methods is exploding. With HPC systems being severely constrained by power/energy, storage, and I/O, it has become very important to gain a thorough understanding of the trade-offs among the various in-situ approaches. Our experience with ETH and traditional full-featured softwares strongly indicate the need for a light-weight mechanism to quickly explore large parameter spaces so as to make informed choices about setting the visualization pipelines at extreme scale. Our initial experiments already point to important directions in designing a visualization workflow in order to save time, power, energy, while still obtaining good-quality visualizations. We expect ETH to expand the type of toolkits available to a domain scientist, by introducing a testing framework in addition to the existing production-oriented visualization frameworks.

Acknowledgment

This material is based upon work supported by Dr. Lucy Nowell of the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research under Award Numbers DE-SC0012513, DE-SC0012637, and DE-SC-0012516. We would like to acknowledge Galen Gisler (LANL) for the asteroid simulation and contributors to the hardware/hybrid accelerated cosmology Code (HACC). This work was published under LANL release LA-UR-17-26715.

References

[1] S. Ahern, A. Shoshani, K.-L. Ma, A. Choudhary, T. Critchlow, S. Klasky, V. Pascucci, J. Ahrens, W. Bethel, H. Childs, et al. Scientific Discovery at the Exascale: Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and visualization. *Dept. of Energy Office of Advanced Scientific Computing Research*, 2011.

[2] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, and J. Mauldin. Paraview Catalyst: Enabling In Situ Data Analysis and Visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pages 25–29. ACM, 2015.

[3] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, and E. W. Bethel. In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. *Computer Graphics Forum*, 35(3):577–597, 2016.

[4] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, et al. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–9. IEEE, 2012.

[5] M. Dorier, R. Sisneros, T. Peterka, G. Antoniu, and D. Semeraro. Damaris/viz: A Nonintrusive, Adaptable and User-friendly In Situ Visualization Framework. In *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, pages 67–75. IEEE, 2013.

[6] A. Esnard, N. Richart, and O. Coulaud. A steering environment for online parallel visualization of legacy parallel simulations. In *Distributed*

Simulation and Real-Time Applications, 2006. DS-RT'06. Tenth IEEE International Symposium on, pages 7–14. IEEE, 2006.

[7] T. Evans, W. L. Barth, J. C. Browne, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra. Comprehensive Resource Use Monitoring for HPC Systems with TACC Stats. In *Proceedings of the First International Workshop on HPC User Support Tools (HUST)*, pages 13–21, 2014.

[8] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. E. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 89–96. IEEE, 2011.

[9] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. E. Jansen. The ParaView Coprocessing Library: A Scalable, General Purpose In-Situ Visualization Library. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 89–96, Oct 2011.

[10] M. Gamell, I. Rodero, M. Parashar, J. C. Bennett, H. Kolla, J. Chen, P.-T. Bremer, A. G. Landge, A. Gyulassy, P. McCormick, et al. Exploring Power Behaviors and Trade-offs of In-situ Data Analytics. In *2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12. IEEE, 2013.

[11] B. Geveci. personal communication.

[12] M. Gittings, R. Weaver, M. Clover, T. Betlach, N. Byrne, R. Coker, E. Dendy, R. Hueckstaedt, K. New, W. R. Oakes, et al. The RAGE Radiation-Hydrodynamic Code. *Computational Science & Discovery*, 1(1):015005, 2008.

[13] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, et al. HACC: Simulating Sky Surveys on State-of-the-Art Supercomputing Architectures. *New Astronomy*, 42:49–65, 2016.

[14] R. Haimes. pv3-a distributed system for large-scale unsteady cfd visualization. In *32nd Aerospace Sciences Meeting and Exhibit*, page 321, 1994.

[15] S. Institute, 2016. SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI), Download from: http://www.scirun.org.

[16] A. Knoll, I. Wald, P. A. Navrátil, M. E. Papka, and K. P. Gaither. Ray tracing and volume rendering large molecular data on multi-core and many-core architectures. In *Proceedings of the 8th International Workshop on Ultrascale Visualization*, UltraVis '13, pages 5:1–5:8, New York, NY, USA, 2013. ACM.

[17] J. A. Kohl, T. Wilde, and D. E. Bernholdt. Cumulvs: Interacting with high-performance scientific simulations, for visualization, steering and fault tolerance. *International Journal of High Performance Computing Applications*, 20(2):255–285, 2006.

[18] T. Kuhlen, R. Pajarola, and K. Zhou. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. 2011.

[19] S. Labasan, M. Larsen, and H. Childs. Exploring tradeoffs between power and performance for a scientific visualization algorithm. In *5th IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 73–80. IEEE, 2015.

[20] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. Samatova. Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data. *Euro-Par 2011 Parallel Processing*, pages 366–379, 2011.

[21] M. Larsen, E. Brugger, H. Childs, J. Eliot, K. Griffin, and C. Harrison. Strawman: A Batch In Situ Visualization and Analysis Infrastructure for Multi-Physics Simulation Codes. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pages 30–35, 2015.

[22] M. Larsen, C. Harrison, J. Kress, D. Pugmire, J. S. Meredith, and H. Childs. Performance modeling of in situ rendering. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 24:1–24:12, Piscataway, NJ, USA, 2016. IEEE Press.

[23] J. Lofstead, R. Oldfield, T. Kordenbrock, and C. Reiss. Extending Scalability of Collective IO through Nessie and Staging. In *Proceedings of the sixth workshop on Parallel Data Storage*, pages 7–12. ACM, 2011.

[24] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pages 15–24. ACM, 2008.

[25] K.-L. Ma. In Situ Visualization at Extreme Scale: Challenges and Opportunities. *IEEE Comput. Graph. Appl.*, 29(6):14–19, Nov. 2009.

[26] U. D. of Energy. Synergistic Challenges in Data-Intensive Science and Exascale Computing. Mar 2013.

[27] S. Parker, M. Parker, Y. Livnat, P. P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):238–250, Jul 1999.

[28] K. Perlin. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 681–682, New York, NY, USA, 2002. ACM.

[29] I. Rodero, M. Parashar, A. G. Landge, S. Kumar, V. Pascucci, and P.-T. Bremer. Evaluation of in-situ analysis strategies at scale for power efficiency and scalability. In *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*, pages 156–164. IEEE, 2016.

[30] W. Schroeder, K. Martin, and B. Lorensen. The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics, 2005.

[31] M. J. Turk, B. D. Smith, J. S. Oishi, S. Skory, S. W. Skillman, T. Abel, and M. L. Norman. yt: A multi-code analysis toolkit for astrophysical simulation data. *The Astrophysical Journal Supplement Series*, 192(1):9, 2010.

[32] D. R. J. A. S. P. Vignesh Adhinarayanan, Wu-chun Feng. Characterizing and modeling energy for extreme-scale in-situ visualization. In *Proceedings of the 31st International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017.

[33] V. Vishwanath, M. Hereld, V. Morozov, and M. E. Papka. Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 19. ACM, 2011.

[34] I. Wald, G. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Gnther, and P. Navratil. Ospray - a cpu ray tracing framework for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):931–940, Jan 2017.

[35] J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, S. Habib, and K. Heitmann. In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. In *Computer Graphics Forum*, volume 30, pages 1151–1160. Wiley Online Library, 2011.

[36] S. Ziegeler, C. Atkins, A. Bauer, and L. Pettey. In situ analysis as a parallel i/o problem. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pages 13–18. ACM, 2015.

[37] S. B. Ziegeler. An I/O Mini-App Dedicated to In Situ Visualization. In *2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pages 29–34, Nov 2016.