Eurographics Symposium on Parallel Graphics and Visualization (2018)
H. Childs, F. Cucchietti (Editors)

VisIt-OSPRay: Toward an Exascale Volume Visualization System

Qi Wu[1], Will Usher[1,2], Steve Petruzza[1], Sidharth Kumar[1], Feng Wang[1], Ingo Wald[2], Valerio Pascucci[1] and Charles D. Hansen[1]

[1]Scientific Computing and Imaging Institute, University of Utah
[2]Intel Corporation

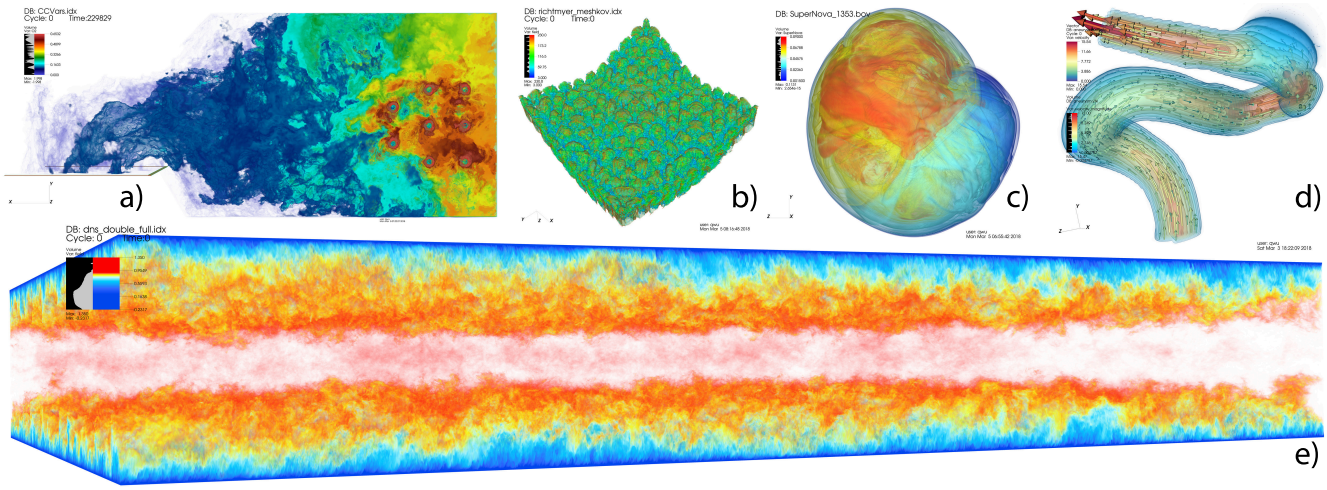

Figure 1: *High-quality interactive volume visualization using VisIt-OSPRay:* ***a)*** *volume rendering of O_2 concentration inside a combustion chamber [CCM18];* ***b)*** *volume rendering of the Richtmyer-Meshkov Instability [CDD*02];* ***c)*** *visualization of a supernova simulation [BM07];* ***d)*** *visualization of the aneurysm dataset using volume rendering and streamlines;* ***e)*** *scalable volume rendering of the 966GB DNS data [MKM99] on 64 Stampede2 Intel® Xeon Phi™ Knight's Landing nodes.*

Abstract

Large-scale simulations can easily produce data in excess of what can be efficiently visualized using production visualization software, making it challenging for scientists to gain insights from the results of these simulations. This trend is expected to grow with exascale. To meet this challenge, and run on the highly parallel hardware being deployed on HPC system, rendering systems in production visualization software must be redesigned to perform well at these new scales and levels of parallelism. In this work, we present VisIt-OSPRay, a high-performance, scalable, hybrid-parallel rendering system in VisIt, using OSPRay and IceT, coupled with PIDX for scalable I/O. We examine the scalability and memory efficiency of this system and investigate further areas for improvement to prepare VisIt for upcoming exascale workloads.

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics;***

1. Introduction

Interactive visualization plays a key role in scientific research, assisting with exploring data, formulating hypotheses, communicating results, and debugging simulations. However, current simulations running on petascale HPC platforms can easily produce datasets beyond what can be visualized on typical workstations, making interactive visualization of such data challenging. To address this challenge, numerous distributed parallel rendering techniques that are capable of achieving interactive framerates have been proposed [Hsu93, HBC12, KWN*13, EBA12, GPC*15]. However, these methods tend to have project-specific implementations, or are not incorporated into common visualization tools, making them inaccessible to general users. General visualization software like ParaView [AGL05] and VisIt [CBW*12] include support for distributed rendering via a client/server architecture; however, their performance can be far behind the current state-of-the-art. As simulations grow in scale, the integration of new scalable rendering techniques into production visualization software is crucial for general science users. By integrating such functionality into existing

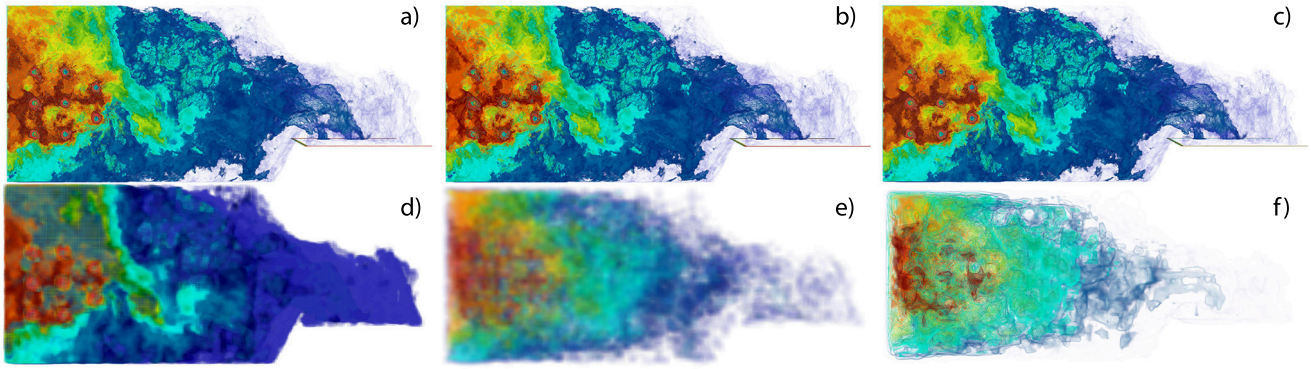

Figure 2: *Volume rendering comparison of six different renderers in VisIt: (a) RayCasting:Compositing; (b) RayCasting:SLIVR; (c) OSPRay; (d) Splatting; (e) 3D-Texture; (f) SLIVR. Our OSPRay integration (c) can produce high-quality images identical to the baseline (a) at up to 30 times higher framerates.*

tools, users who are familiar with these widely used systems will benefit from scalable parallel rendering, without having to learn new, project-specific, visualization systems to explore their data interactively.

In this work, we present VisIt-OSPRay, a scalable volume visualization system integrated into VisIt, which focuses on addressing challenges encountered when visualizing hundred gigabyte- to terabyte-sized volumetric datasets on recent Intel® Xeon® and Intel® Xeon Phi™ processor-based architectures. One such motivating dataset is the coal-boiler simulation produced by the University of Utah's Carbon-Capture Multidisciplinary Simulation Center (CCMSC) [CCM18] using the Uintah Computational Framework [PGH06]. This simulation aims to guide the design of next-generation electric power plants by integrating simulation into the design process. Although such simulations can already produce up to petabytes of data, simulating the coal-boiler at the resolution and accuracy required to accurately evaluate designs gives rise to problems between 50 to 1000 times larger than those solvable on current machines, making such problems perfect candidates for exascale computing.

To enable interactive visualization of large-volume datasets in VisIt and leverage shared-memory parallelism on current and emerging CPU and many-core architectures, we re-engineered VisIt's distributed renderer to a hybrid MPI + threads model. On each node, we use OSPRay [WJA*17] for fast CPU-based rendering. To improve image compositing, we use IceT [MKPH11] inside VisIt's sort-last rendering pipeline, instead of the direct-send compositor used by VisIt's *RayCasting:SLIVR* renderer. Finally, for fast parallel I/O, we use the Parallel IDX (PIDX) [KVC*11] library.

Our implementation achieves up to 30 times higher framerates than VisIt's *RayCasting:Compositing* renderer, while producing equivalent high-quality images (Figure 2). Furthermore, we report significant strong and weak scaling improvements up to 32,768 CPU cores on recent HPC platforms equipped with Intel® Xeon Phi™ (KNL) processors and Intel® Xeon® Platinum Skylake (Skylake) processors. Our specific engineering contributions to VisIt's distributed rendering capabilities are:

- Moving to a hybrid-parallel execution model,
- Leveraging OSPRay for fast volume rendering, and
- Improving the use of IceT for image compositing.

2. Previous Work

Volume rendering is a standard approach for visualizing 3D scalar fields, first proposed in the late 80s [Lev88, DCH88, Sab88]. Although later work by Levoy [Lev90] improved the performance of ray-tracing-based methods by introducing empty space skipping and early ray termination, volume rendering remains a computation-, memory-, and I/O-intensive task. A large body of work has continued to study how the performance can be improved further.

In the context of desktop volume rendering, previous work has investigated the use of GPUs [CN93, CCF94, KW03, MRH10] and better leveraging CPUs [KTW*11, RWCB15, WJA*17]. Methods such as rendering proxy geometry [KW03] and direct volume ray-tracing [MRH10] have been used extensively, as both can be easily parallelized. In this work, we use OSPRay [WJA*17] as the rendering back-end, which is a state-of-the-art high-fidelity CPU ray-tracing library for interactive scientific visualization. It is implemented using Embree [WWB*14], for basic high-performance ray-tracing kernels and acceleration structures, and ISPC [PM12], to write well-vectorized code when implementing renderers, volumes, and geometries in the ray tracer. Moreover, OSPRay offers an easy-to-use C API, is extendable via modules (e.g., [WKJ*15, VSW*17, UAB*17]), and is open-source.

In the field of distributed volume rendering, extensive research has focused on reducing image compositing time for sort-last rendering [MEFC94] on large parallel systems. Although the volume rendering process is easily parallelized, the final compositing step amounts to a global reduction and becomes the main performance bottleneck at high core counts. A great number of image compositing algorithms can be found in the literature. See, for example, serial direct send [Hsu93, MPHK93], binary tree [Kui91], binary-swap [MPHK93, YCM08], and radix-k [PGR*09], all of which are available in IceT [MKPH11]. These methods typically consider only MPI-parallel rendering, where one process is run per

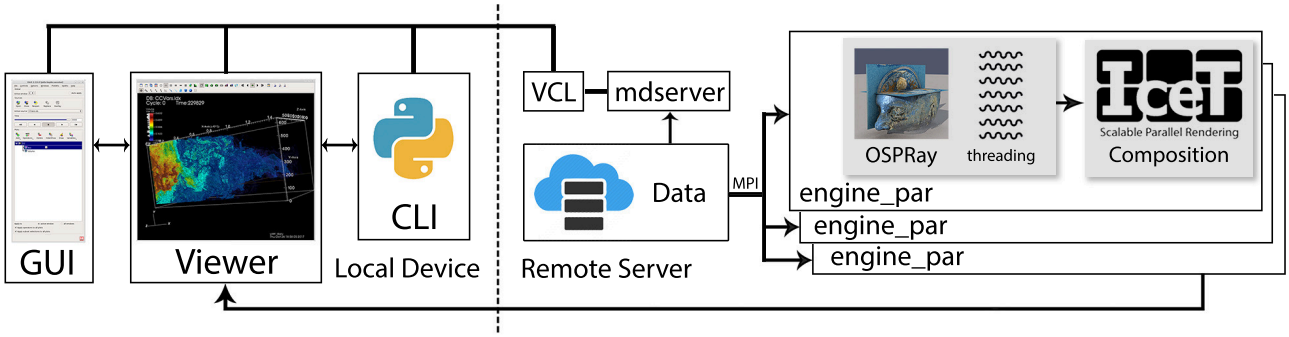

Figure 3: *The high-level component-based design of VisIt's client server system. The GUI, Command Line Interface (CLI) and Viewer components run on the user's workstation and are in charge of handling user interactions and displaying the visualization. Updates to the visualization and data flow are sent to the VisIt Component Launcher (VCL) component running on the remote server. On the remote server, the VCL will launch the mdserver, responsible for metadata retrieval, on the master node, and multiple parallel engines on the compute nodes, to execute computation and rendering tasks. The partial images produced by each engine will then be composited (e.g., using IceT) to produce the final image that is sent back to the viewer.*

core. Howison et al. [HBC12] demonstrated that for multi- and many-core CPUs, hybrid parallelism—specifically, using threads and shared memory for intra-node parallelism and MPI for inter-node communication—is more efficient in terms of speed and memory than using only MPI for both intra- and inter-node parallelism.

Grosset et al. [GPC*15] proposed the TOD-Tree algorithm, which uses parallel direct send for intra-node compositing and a K-ary tree method for compositing between nodes. The TOD-tree algorithm maps well to multi- and many-core CPUs by using threads for parallel direct send within a node and MPI for tree-based compositing between nodes. However, TOD-Tree requires that the volume bricks owned by each node together form a convex shape, precluding it from use in VisIt, which cannot make such a guarantee. In VisIt, the volume bricks assigned to each node are often not convex in aggregate, due to the use of load balancing.

Prior studies with similar goals to our own have also sought to integrate scalable volume rendering into production visualization software. Childs et al. [CDM06] implemented the first parallel volume renderer in VisIt, *RayCast:Compositing*, using a sort-middle pipeline, providing the ability to render large data at full resolution to VisIt users. Fogal et al. [FCS*10] developed a data-distributed multi-GPU volume rendering system in VisIt that supported static and dynamic load balancing via a K-D tree decomposition. Furthermore, Fogal et al. [FCS*10] implemented a data bypassing scheme to prevent VisIt from down-sampling the volume data when it exceeded a single GPU's memory and integrated IceT for efficient image compositing. To accelerate software rendering, Brownlee et al. [BPL*12] integrated Manta [BSP06], a fast CPU ray tracer, into both VisIt and ParaView. This integration was further improved by Brownlee et al. [BFH12] with GluRay, a library that redirects OpenGL calls to Manta for efficient rendering. However, these implementations (Manta, GluRay) were intended for use on a single workstation, rather than distributed rendering on clusters.

The Remote Visualization Group at the Texas Advanced Computing Center (TACC) integrated OSPRay into ParaView and VisIt via two libraries called *pvOSPRay* and *visitOSPRay*. Their integration mapped OSPRay's API directly to VTK [SLM04], making the OSPRay integration transparent to the rest of the system, at the expense of adding an additional layer of complexity. To further improve performance and reduce complexity, the ParaView team has replaced *pvOSPRay* by directly integrating OSPRay as a VTK render pass. In our work, we follow a similar approach and integrate OSPRay as an `avtFilter` in VisIt.

3. Volume Rendering in VisIt

As a general tool for visualization, VisIt supports several volume rendering algorithms. These rendering algorithms can be classified into two categories, describing whether or not the algorithm is hardware accelerated. Hardware-accelerated renderers in VisIt, e.g., the *Splatting* [Wes90] and *3D Texture* [CN93] renderers, leverage acceleration devices such as GPUs for rendering. When rendering with a hardware-accelerated method VisIt will down-sample the volume data to fit it into the GPU memory, precluding such methods from creating high-quality images of large datasets using distributed rendering (see Figure 2). There is another ongoing project in parallel with our work to integrate OSPRay into VisIt [Hot18], which proposes to integrate OSPRay as an acceleration device, thus precluding it from rendering large datasets at high quality.

Software renderers, also referred to as scalable renderers (SR) by VisIt, can provide parallel data-distributed rendering using CPUs. VisIt currently has two software renderers: *RayCasting:Compositing* and *RayCasting:SLIVR*. The *RayCasting:Compositing* renderer [CDM06] employs a sort-middle approach and computes samples by sending rays between nodes as they traverse the distributed volume. After the rays have sampled the volume, the computed samples are composited in order to create the final image. This method is inefficient due to its huge memory footprint and inter-node communication requirements. The *RayCasting:SLIVR* renderer implements a typical sort-last rendering

pipeline using parallel direct send [Hsu93]. *RayCasting:SLIVR* has been found to be faster than *RayCasting:Compositing*, as it requires significantly less inter-node communication; however, it is still inefficient on recent many-core processors in terms of memory footprint and overall performance.

4. Implementation

To provide context for our implementation within VisIt, we first describe the VisIt rendering pipeline (Section 4.1) and then our use of PIDX for fast I/O (Section 4.2). Finally, we present the details of our OSPRay integration (Section 4.3) and our approach to improving compositing performance by better utilizing IceT in a hybrid-parallel renderer (Section 4.4).

4.1. The VisIt Distributed Rendering Pipeline

VisIt employs a client-server architecture for distributed visualization (Figure 3), where user instructions and data processing and rendering states are gathered by components running on the user's workstation and sent to the VisIt Component Launcher (VCL) on a remote server. Data processing and scalable rendering tasks are executed by multiple parallel compute engines (engine_par) launched by the VCL on the compute nodes. The compute engines communicate with each other via MPI. The Analysis and Visualization Toolkit (AVT) is used by VisIt to set up the data processing flow through an analysis pipeline (Figure 4). A typical AVT pipeline consists of a data *source*, e.g., a file reader; several *filter*s, which transform and process the data; and finally, a data *sink*, e.g., a renderer to display the result. A volume renderer can be implemented in VisIt as either a sink component, which directly displays pixels to the screen (hardware-accelerated methods), or a filter component, which transforms the 3D grid into an image (scalable rendering methods). Only the latter approach allows for data-distributed rendering, and we therefore implement our volume renderer as a filter.

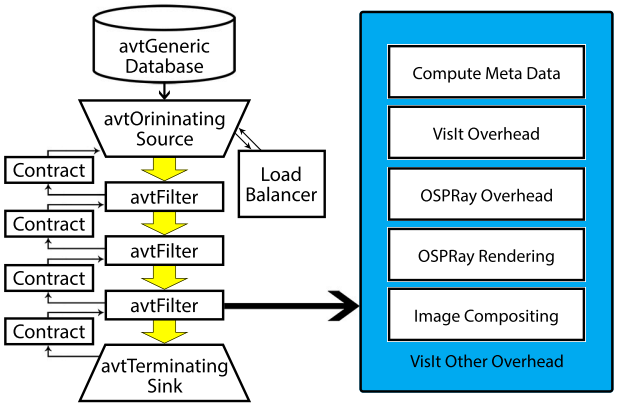

Figure 4: *Left: the AVT data processing pipeline in VisIt. Right: the six operations performed in the* `avtRayTracer` *filter. In "Compute Meta Data", coordinates are transformed to camera space. They are then passed to VisIt and OSPRay in "VisIt Overhead" and "OSPRay Overhead" stages. In the "OSPRay Rendering" and "Image Compositing" stages, the volume is rendered, and the image is returned to VisIt. Any other operations are classified as "VisIt Other Overhead".*

4.2. Data I/O via PIDX

Post-processing visualization tasks are generally performed using fewer resources (e.g., memory and cores) compared to the simulation that produced the data. Therefore, post-processing visualization requires readers to be able to fetch large amounts of data efficiently in very different scenarios. For example, many data formats (e.g., BOV or Uintah's UDA) store data in bricks or patches, which turn into many small files on disk. As a result, the reader must execute numerous small accesses to the filesystem, which negatively impacts I/O performance. The PIDX library [KVC*11] instead allows for full control over the number and size of output files, through a two-phase I/O process, where data is aggregated onto a subset of the cores before being saved to the disk. The Uintah Computational Framework utilizes PIDX for checkpoint-restart due to the superior I/O performance over previous I/O methods. Moreover, the library is able to quickly fetch arbitrary blocks of data, independent of the initial domain decomposition used to write the file.

These features enable visualization tasks to control the amount of data assigned to each processor or node. For example, volume rendering tasks are no longer required to render volume bricks using the bricking selected by the simulation, and can instead load the data with a more favorable distribution for rendering. In our PIDX reader for VisIt, we compute a domain decomposition that assigns one convex brick of data to each node, where the bricks divide the domain as uniformly as possible. This ability to change the domain decomposition enables us to adopt a hybrid-parallel rendering approach in the style of TOD-Tree [GPC*15] to better utilize multi- and many- core CPUs. For example, if we load a data using 128 Intel® Xeon Phi™ processors, we need to launch only 128 program instances and divide the data into 128 domains, while the program can still utilize all 8192 cores via multi-threading.

4.3. Rendering via OSPRay

Our OSPRay rendering filter maps features provided by the VisIt *RayCasting:Compositing* renderer to OSPRay, allowing users familiar with the previous renderer to easily produce the same images, albeit at much higher framerates. To select our renderer, users can simply choose the *RayCasting:OSPRay* rendering option in the GUI. We now describe our mapping from VisIt's rendering parameters and data representation to OSPRay's.

Camera Matrix Transformations

VisIt defines its camera transforms following OpenGL conventions; however, OSPRay is a ray tracer and generates primary rays differently. Although the methods for specifying the camera transform differ, there is an easy mapping between most parameters of VisIt's camera to OSPRay's. This mapping can be implemented in VisIt directly, or by extending OSPRay's camera. We chose to implement this mapping in VisIt. In our implementation, we found only three

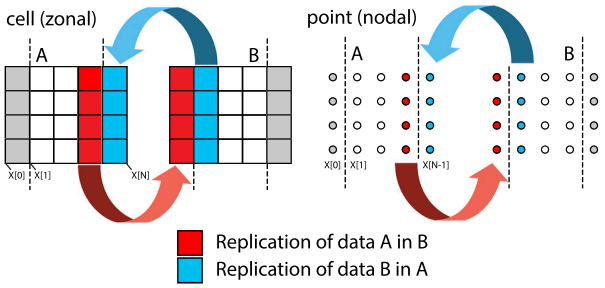

Figure 5: *Volume and ghost region representation in VisIt.* `X[i]` *indicate the cell boundaries for zonal data and point positions for nodal data. If two bricks share a boundary (e.g., A and B), there will be a layer of replicated voxels on each node at the boundary, to allow for correct interpolation. To hide the ghost voxels on each brick, VisIt-OSPRay applies clipping planes at the cell boundaries for zonal data and at the mid-point between adjacent points for nodal data.*

parameters that required special treatment to produce an equivalent rendering.

In VisIt, the `imagePan` and `imageZoom` parameters define screen-space translation and zoom operations, which are essential for user-interactions. To match the effect of these parameters in our OSPRay renderer, we map them to the `imageStart` and `imageEnd` parameters of OSPRay's camera, which are used to describe the region of the camera sensor being rendered. VisIt also allows users to perform arbitrary scaling along the XYZ axes using the `axis3DScales` parameter. This operation is equivalent to scaling the volume grid cell size along different axes. We map the scaling applied to the `gridSpacing` parameter on the volume in OSPRay, which similarly allows users to specify a scaling to apply to the volume grid.

Volume Representation

VisIt uses VTK [SLM04] to represent various datasets used in scientific visualization. The `vtkRectilinearGrid` is used to represent regular 3D grids. Depending on whether the volume has data at every point, volumes are classified as either nodal or zonal. Zonal data (Figure 5 left) uses the bottom-left corner of the cell to represent the voxel position. Nodal data (Figure 5 right) uses the node position as the voxel position. In order to achieve data-distributed rendering, we extended OSPRay with a module, *module_visit*. Although a data-distributed rendering mode has recently been included in the official OSPRay, it was not available when we started our integration, and therefore we kept our implementation, which is still compliant with the current OSPRay system.

To preserve the appearance of a single connected volume in data-distributed rendering, ghost voxels must be introduced along shared boundaries for correct interpolation across the boundary [Hsu93]. Although VisIt already manages the ghost voxel exchange between ranks (blue voxels for A, red voxels for B, Figure 5), OSPRay's volume types have no concept of ghost voxels, and treat them as regular voxels owned by the node, resulting in them being incorrectly rendered. To hide the

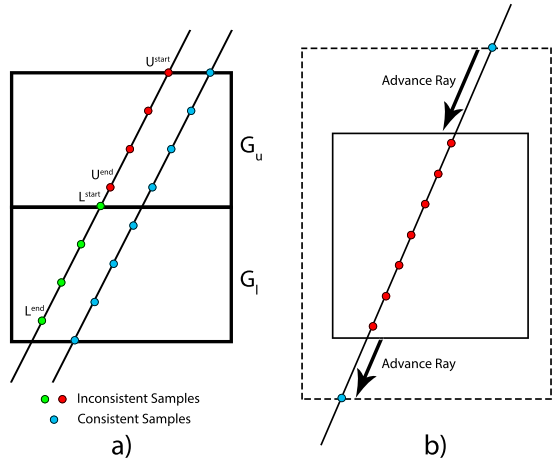

Figure 6: *a) Differences between inconsistent sampling and consistent sampling. For the ray on the left, blending the red and green samples together will be incorrect, as the last red sample U^{end} and the first green sample L^{start} are too close to each other. b) In VisIt-OSPRay, different bricks have an identical global volume bounding box, and rays are adjusted in order to produce correct samples.*

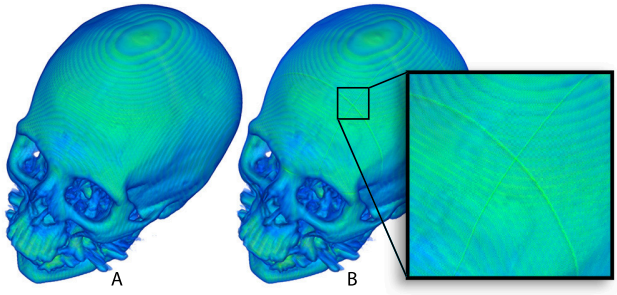

Figure 7: *The rendering artifact produced by inconsistent sampling between bricks. A) The CT-head dataset rendered by VisIt-OSPRay using consistent sampling between bricks; B) the same data rendered using OSPRay's built-in volume type, resulting in discontinuities at the brick boundaries.*

ghost voxels we can use the `volumeClippingBoxLower` and `volumeClippingBoxUpper` parameters of OSPRay's volume types to clip them, while still leaving them available in the data for correct interpolation between boundaries. The dashed lines in Figure 5 show where the clipping planes are placed.

It is well known that having consistent sampling steps across bricks is important to avoid artifacts in distributed volume rendering [Hsu93]. Consider a renderer running on two ranks, R_u and R_l, as shown in Figure 6a. The volume is divided into two bricks, G_u and G_l, with one brick assigned to each rank. If both R_u and R_l sample their brick starting from the ray's intersection with their local bounds, there will be a different sampling step size between the last sample of R_u and the first sample of R_l (red and green samples, Figure 6a). The final composited color produced by these samples will be incorrect when compared to sampling a single continuous

volume (blue samples, Figure 6a), resulting in visible artifacts (Figure 7).

To ensure consistent sampling between different bricks, we extended OSPRay with a new OSPRay volume type, the `visit_shared_structured_volume`. This new volume solves the sampling consistency issue by using the concept of a "global volume". Specifically, while each renderer holds data for just its local brick, the volume bounds reported to OSPRay's renderer are those of the global volume. When OSPRay computes a ray intersection with the volume, it will intersect the ray with the full volume bounds on each node and begin sampling at the same first hit point (Figure 6b). To compute samples, OSPRay calls into our sample method, which will skip the ray forward to the first sample point within the local brick, sample the brick, and then find the ray's exit point from the global volume. This method requires no additional communication between nodes, because each node must be aware of only the global dimensions, which are available as meta-data in the volume file. Table 1 shows how to set parameters for this new volume type.

Transfer Functions

OSPRay currently supports only 1D piecewise-linear transfer functions; however, VisIt provides multiple options, including Gaussian transfer functions. To map VisIt's transfer function to OSPRay, we resample the transfer function to a 1K element array to produce a piecewise-linear approximation. Furthermore, we removed VisIt's hard-coded opacity correction term, $1-(1-\alpha)\frac{\Delta x}{\Delta x'}$, which it uses to account for the sampling rate, because this correction is handled automatically by OSPRay.

Combining OSPRay and VisIt Renderings

One of the advantages of general purpose visualization software is that it allows users to combine multiple visualization modalities, e.g., different geometries such as streamlines or isosufaces and volumes into one image. To support combining geometry rendered by VisIt with distributed volume data rendered with OSPRay, we take advantage of the the `maxDepthTexture` parameter in the OSPRay SciVis Renderer. This parameter allows us to pass a previously produced depth image (e.g., from OpenGL) to OSPRay, which will be used to terminate rays correctly against the previously rendered geometry. Figure 8 shows two examples of combining our OSPRay volume rendering with VisIt-rendered geometry.

4.4. Parallel Image Compositing with IceT

It has been shown that in data-parallel rendering at large core counts, the final image compositing step becomes the main bottleneck [GPC*15]. This bottleneck can be addressed by moving to a hybrid-parallel model [GPC*15], where compositing is done within a node using threads and shared memory, and between nodes with MPI. In our renderer, we partially adopt the TOD-Tree algorithm of Grosset et al. [GPC*15]. Specifically, we employ TOD-Tree's approach for multi-threading for intra-node compositing, and we extend this approach to leverage vectorization.

In the case that each node contains only one brick, and thus no intra-node compositing is needed, we use IceT [MKPH11] for inter-node compositing. IceT provides a wide range of compositing algorithms, such as reduce, radix-k, binary-swap, and tree-like compositing. In our experiments (Section 5), we found that the cluster network topology can have a significant influence on compositing performance. On a fat-tree topology network, we found the tree-like compositing approach outperformed the other algorithms available in IceT. Since the fat-tree topology is relatively common, we selected the tree-like compositing approach as the default method.

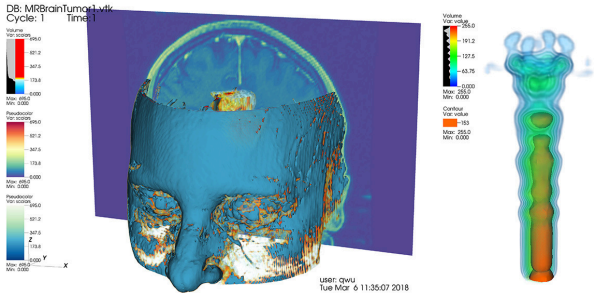

Figure 8: *Displaying volumes with geometry rendered by VisIt. Left: the MRI Brain Tumor dataset visualized using a volume, a slice, and an isosurface. Right: The Fuel dataset visualized with a volume and isosurface.*

5. Results

We evaluated three key aspects of our VisIt-OSPRay integration: the scalability of the OSPRay rendering component and improved image compositing (Section 5.2); the overall performance improvement achieved in Visit as a whole (Section 5.3); and finally, the memory consumption of VisIt-OSPRay relative to other renderers in VisIt (Section 5.4).

The benchmarks were performed on the *Stampede2* supercomputer at the Texas Advanced Computing Center (TACC) and the *Theta* supercomputer at Argonne National Laboratory (ANL). *Stampede2* uses an Intel® Omni-Path network with a fat-tree topology employing six core switches; it has 4,200 Intel® Xeon Phi™ 7250 (KNL) processor-based nodes and 1,736 Intel® Xeon® Platinum 8160 (Skylake) processor-based nodes. Each KNL node has 96 GB RAM and 68 cores. Each Skylake node has 192 GB RAM and 48 cores over two sockets. *Theta* is a Cray XC40 machine with an Aries interconnect in a Dragonfly configuration. *Theta* is equipped with 4,392 KNL 7230 nodes, each with 64 cores and a 96 GB RAM. We ran additional benchmarks on two Intel® Xeon® E5 (Haswell) processor-based clusters: *Cooley* and *Kepler*. *Cooley* is the visualization cluster at ANL, with 126 twelve-core nodes. *Kepler* is located at the Scientific Computing and Imaging Institute (SCI) at the University of Utah and has 32 sixteen-core nodes.

We selected different compilers and libraries on *Stampede2* and *Theta*. On *Stampede2*, we used the Intel® compiler version 17 and Intel® MPI, and on *Theta* we used GCC 5.4 and Cray MPICH, due to some issues encountered when compiling with the Intel compiler. As OSPRay uses threads internally for rendering, we ran a single rendering process per node to achieve the best ren-

Type	Name	Default	Description
bool	`useGridAccelerator`	True	Whether the empty space skipping grid will be built
vec3	`volumeGlobalBoundingBoxLower`	disabled	Lower coordinate of the global volume
vec3	`volumeGlobalBoundingBoxUpper`	disabled	Upper coordinate of the global volume

Table 1: *Additional parameters for the* `visit_shared_structured_volume`.

dering performance. However, for a fair comparison against previous renderers in VisIt that do not support multi-threading, we ran one VisIt renderer per core on each node when benchmarking these renderers. On *Stampede2*, due to memory limitations on the KNL nodes, we could use only 64 of the 68 cores when benchmarking these renderers.

5.1. Dataset

We evaluated VisIt-OSPRay on datasets ranging in size from small (e.g., the supernova simulation [BM07] and aneurysm datasets, Figure 1c,d), medium (e.g., the Richtmyer-Meshkov Instability [CDD*02] Figure 1b), large (e.g., the coal-boiler Figure 1a), to extremely large (e.g., the DNS [MKM99] Figure 1e). Our project was initially motivated by the challenges encountered by the University of Utah's Carbon-Capture Multidisciplinary Simulation Center (CCMSC) when trying to visualize their coal-boiler dataset. The coal-boiler consists of 1811 timesteps, each with 50 fields at a resolution of $1466 \times 648 \times 578$, modeling the evolution of an ultra super-critical coal boiler powerplant. Due to the dataset's massive size (~179 TB total), the CCMSC has used VisIt extensively for visualization and analysis tasks.

While the coal-boiler is large in aggregate, each individual field is not. To evaluate our system on an extremely large single volume, we performed additional benchmarks on the DNS dataset [MKM99]. The DNS, produced by the Institute for Computational Engineering and Science (ICES) at the University of Texas-Austin, is a single $10240 \times 7680 \times 1536$ double-precision volume (966 GB) simulating turbulent flows.

For strong scaling benchmarks, volumes were rendered to a 1024×933 framebuffer. For weak scaling runs, volumes were rendered to a $1024\sqrt{N} \times 1024\sqrt{N}$ framebuffer, where N represents the number of nodes. We measured rendering performance by taking the average framerate while rotating the camera along the X and Y axes. The first frame was not included for all cases, because it includes executing additional setup overhead in VisIt, which usually takes two to three times longer than subsequent frames.

5.2. Scalability

We evaluated the strong scaling of VisIt-OSPRay's core rendering component by measuring the average time spent in `ospRenderFrame` (Figure 9). We found that OSPRay performed very well on strong scaling and can effectively utilize recent multi- and many-core processors, such as the Intel® Xeon® processors (Skylake) on *Stampede2* and the Intel® Xeon Phi™ processors (KNL) on *Stampede2* and *Theta*, for both medium (the coal-boiler) and large (the DNS) datasets.

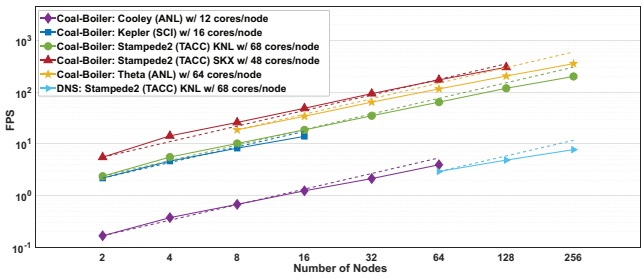

Figure 9: *Strong scaling of our OSPRay rendering component on different machines. The dashed lines indicate the ideal strong scaling trend.*

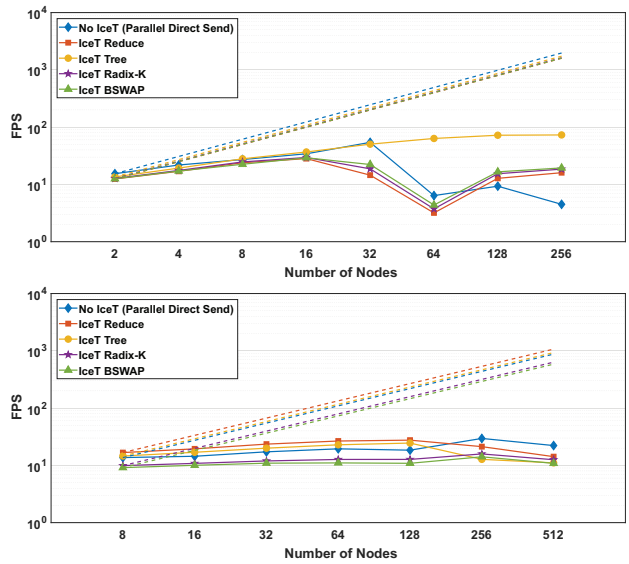

Figure 10: *Strong scaling on the coal-boiler dataset using image compositing algorithms available in IceT and VisIt. The benchmarks were run on Stampede2 KNLs (top) and Theta (bottom).*

We also studied the impact of different compositing strategies on the scalability of VisIt-OSPRay's image compositing component. We compared four compositing algorithms available in IceT: reduce, binary-tree, radix-k, and binary swap; and the parallel direct send algorithm in VisIt's *RayCasting:SLIVR* renderer on *Stampede2* and *Theta* (Figure 10). Although the rendering component discussed above is independent across nodes and would be expected to scale well, image compositing requires communica-

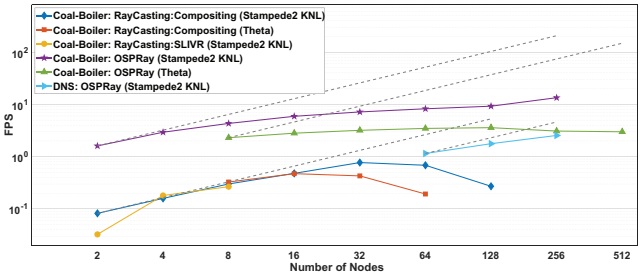

Figure 11: *Strong scaling of the overall framerate of VisIt-OSPRay compared to the RayCasting:Compositing and RayCasting:SLIVR renderers. OSPRay is run with multiple threads and one process per node; the other renderers are run with 64 processes on each node. Dashed lines indicate ideal strong scaling. We also included the strong scaling result on the DNS dataset (cyan).*

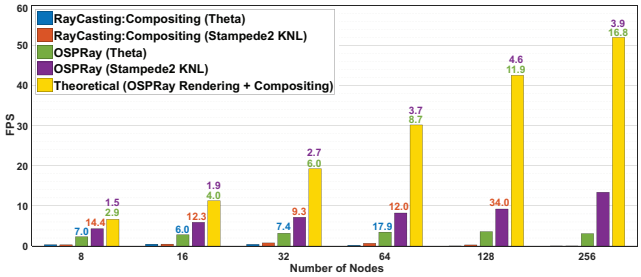

Figure 12: *The overall speed-up we achieved with VisIt-OSPRay (green, purple) and the theoretical framerate we could reach (yellow) if the rest of the software pipeline was perfectly optimized. The numbers on bars indicate their speed-ups and the colors indicate the renderers being compared. For example, the green numbers on the yellow bars represent the speed-up of VisIt-OSPRay's theoretical performance in comparison with the actual performance we measured on Theta. Although our integration of OSPRay has yielded significant improvement, additional overheads in VisIt remain that impede performance. The coal-boiler dataset was used for these benchmarks.*

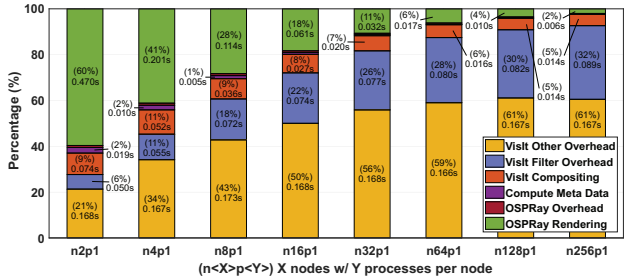

Figure 13: *The performance profile of VisIt-OSPRay on Stampede2 KNL nodes using the coal-boiler dataset. We find that the time spent rendering with OSPRay rendering (green) and compositing with IceT (orange) together takes only 7% of the total frame time at 256 nodes.*

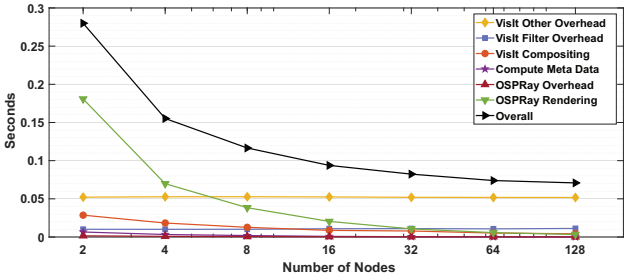

Figure 14: *Strong scaling using the coal-boiler on Stampede2 Skylake nodes broken down by stage. We find that "VisIt Other Overhead" does not scale with the rest of the rendering pipeline.*

tion and synchronization between nodes, as it amounts to a global reduction operation.

On *Stampede2* (Figure 10 top), we observed that when running on fewer than 28 nodes, the number of nodes connected by a leaf switch, the compositing strategies all performed similarly. At 32 nodes, we found a decrease in performance with most algorithms, and as we scaled further to 256 nodes, we found that all the methods besides IceT's tree algorithm perform worse than at 16 nodes. Interestingly, we observed the worst performance for most methods to occur at 64 nodes, which we believe is associated with the network configuration of *Stampede2*. However, we have not run more detailed experiments at this time to investigate the issue.

On *Theta*, however, we found unsatisfying strong scaling for all the image compositing strategies tested. The difference in scalability found when comparing *Stampede2* and *Theta* is likely attributable to the different network topologies and job scheduling strategies employed. *Stampede2*'s network uses a typical fat-tree topology, whereas *Theta* uses a Dragonfly topology. These topologies can have strong effects on communication between nodes, depending on locality and communication patterns. Moreover, *Stampede2* uses SLURM to schedule jobs (which supports topology-aware resource allocation), while *Theta* uses Cobalt (which employs a random task-to-core mapping by default). The random mapping used by Cobalt can result in poor communication locality for communication-heavy tasks like image compositing.

5.3. Overall Performance

To evaluate how much performance improvement our integration of OSPRay can bring to VisIt's rendering system as a whole, we also measured strong scaling of the average framerate achieved in VisIt when using our renderer (Figure 11). The overall strong scaling trend of our OSPRay integration was similar to what we observed in Figure 10, and we achieved up to an order of magnitude speed-up (6–34 times) compared to VisIt's current renderers. However, we also observed that the absolute framerates achieved with our integration are lower than what we would expect, given the framerates observed in Figure 10.

When comparing just the rendering and compositing time (i.e., our theoretically achievable framerate) against the actual framerate achieved in VisIt (Figure 12), we found a significant performance bottleneck elsewhere in VisIt's rendering pipeline. To determine the

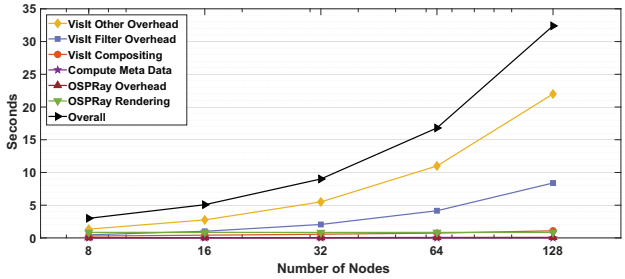

Figure 15: *Weak scaling using the coal-boiler on Theta broken down by stage. We find that "VisIt Other Overhead" and "VisIt Filter Overhead" exhibit poor weak scaling.*

source of this bottleneck, we measured the percentage of time spent in each stage of the rendering pipeline (Figure 13). We also broke down the absolute timing on strong (*Stampede2*) and weak (*Theta*) scaling benchmarks (Figures 14 and 15).

In Figure 13, we found that due to the significant speed-ups we achieved using OSPRay and IceT, the rendering and image compositing stages together—usually considered the major challenges for distributed rendering—accounted for only 7% of the total time in the entire rendering pipeline at 256 nodes. At lower core counts we observe a similar trend, where other software overhead in VisIt starts to dominate the rendering time at eight nodes and up. When comparing the strong and weak scaling of the individual components (Figures 14 and 15) on an absolute scale, we observed that, although the OSPRay rendering and image compositing components scaled relatively well, other stages such as "VisIt Other Overhead" (which sets system parameters and blends the rendered image to the annotated framebuffer on the master rank) did not. At higher node counts, these overheads dominated the overall render time, impacting performance.

5.4. Memory Efficiency

Finally, we examined the impact of our OSPRay integration and migration to hybrid-parallel execution on memory consumption. When rendering large datasets, the additional memory consumed by the renderer and other system components impacts how much data can fit on a node, and thus how many nodes users need to render their data. We profiled the *engine* component, which is responsible for performing the data loading and rendering tasks, using the Linux tool `/usr/bin/time`. We measured the maximum resident memory size required by the program and found that the previous MPI-only renderers required one to two orders of magnitude more memory than our hybrid-parallel OSPRay renderer (Figure 16).

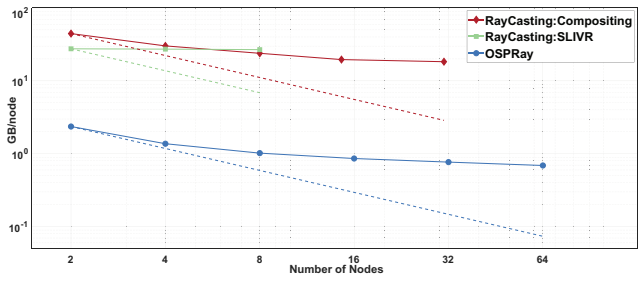

Figure 16: *Comparison of the memory footprint when rendering the coal-boiler of VisIt's* RayCasting:Compositing *and* RayCasting:SLIVR *renderers with our OSPRay renderer on Stampede2 KNL nodes. We find that moving to the hybrid-parallel model of OSPRay comes with a significant reduction in memory use.*

6. Conclusion

In this paper, we have proposed VisIt-OSPRay, an extension of the data-distributed rendering framework in VisIt that aims at moving VisIt further along the path to exascale by significantly improving both absolute performance and scalability on modern Intel® Xeon® and Intel® Xeon Phi™ architectures. Our framework focuses on a hybrid-parallel model that combines multi-threaded rendering in OSPRay with data-parallel rendering in VisIt, with various optimizations such as faster parallel compositing via IceT and more efficient parallel I/O using PIDX.

By leveraging hybrid parallelism, our framework consumes up to an order of magnitude less memory than previous renderers in VisIt; and the improvements in parallel rendering allowed us to scale to up to 32,768 cores across 512 KNL nodes. Combined with the improved single-node rendering performance using OSPRay, we have significantly improved the overall performance and scalability of rendering in VisIt, allowing us to interactively render massive datasets such as the ~179TB in aggregate coal-boiler and the 966 GB DNS.

Perhaps most importantly, our optimizations to VisIt's parallel rendering pipeline have so significantly reduced the cost of rendering, which was once by far the single dominating bottleneck, that rendering is now—at least at scale—merely one of many different costs, with the "real" bottlenecks remaining to be solved for exascale occurring elsewhere in VisIt's pipeline.

7. Future Work

Although we have found that IceT's tree-like compositing strategy can improve image compositing on fat-tree network topology systems, developing a general method to adaptively optimize IceT's compositing strategies remains challenging. In addition, rather than using IceT for compositing, it would also be interesting to evaluate OSPRay's own data-parallel rendering. Although it has never been tested at such scales, its ability to further interleave rendering and compositing might improve scalability even further. Moreover, having a much faster rendering pipeline is also interesting for *in situ* processing. As computational simulations are moving to exascale, *in situ* visualization will become increasingly important, and a closer integration between VisIt-OSPRay and VisIt's LibSim [WFM11] library would be valuable.

Finally, it is imperative to start addressing the non-rendering bottlenecks in VisIt which remain in the way of reaching the performance and efficiency needed for exascale visualization workloads.

It will be interesting to learn how hard—or easy—it will be to parallelize these remaining sequential components to enable exascale visualization and analysis.

Acknowledgements

This research was supported by the DOE, NNSA, Award DE-NA0002375: (PSAAP) Carbon-Capture Multidisciplinary Simulation Center, the DOE SciDAC Institute of Scalable Data Management Analysis and Visualization DOE DE-SC0007446, NSF ACI-1339881, and NSF IIS-1162013. Additional support comes from the Intel Parallel Computing Centers program.

The authors acknowledge the Texas Advanced Computing Center at UT-Austin, and the Argonne Leadership Computing Facility, a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357, for providing HPC resources.

References

[AGL05] AHRENS J. P., GEVECI B., LAW C.: Paraview: An End-User Tool for Large Data Visualization. In *Visualization Handbook*. 2005. 1

[BFH12] BROWNLEE C., FOGAL T., HANSEN C. D.: GLuRay: Ray Tracing in Scientific Visualization Applications using OpenGL Interception. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012). 3

[BM07] BLONDIN J. M., MEZZACAPPA A.: Pulsar Spins from an Instability in the Accretion Shock of Supernovae. *Nature* (2007). 1, 7

[BPL*12] BROWNLEE C., PATCHETT J., LO L.-T., DEMARLE D., MITCHELL C., AHRENS J., HANSEN C. D.: A Study of Ray Tracing Large-scale Scientific Data in Parallel Visualization Applications. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012). 3

[BSP06] BIGLER J., STEPHENS A., PARKER S. G.: Design for Parallel Interactive Ray Tracing Systems. *IEEE Symposium Interactive Ray Tracing* (2006). 3

[CBW*12] CHILDS H., BRUGGER E., WHITLOCK B., MEREDITH J., AHERN S., PUGMIRE D., BIAGAS K., MILLER M., HARRISON C., WEBER G. H., KRISHNAN H., FOGAL T., SANDERSON A., GARTH C., BETHEL E. W., CAMP D., RÜBEL O., DURANT M., FAVRE J., NÁVRATIL P.: VisIt: An End-User Tool for Visualizing and Analyzing Very Large Data. *High Performance Visualization* (2012). 1

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated Volume Rendering and Tomographic Reconstruction using Texture Mapping Hardware. In *IEEE Symposium on Volume Visualization* (1994). 2

[CCM18] CCMSC, UNIVERSITY OF UTAH: The Carbon-Capture Multidisciplinary Simulation Center, 2018. URL: https://goo.gl/YastC6. 1, 2

[CDD*02] COHEN R. H., DANNEVIK W. P., DIMITS A. M., ELIASON D. E., MIRIN A. A., ZHOU Y., PORTER D. H., WOODWARD P. R.: Three-Dimensional Simulation of a Richtmyer-Meshkov Instability with a Two-Scale Initial Perturbation. *Physics of Fluids* (2002). 1, 7

[CDM06] CHILDS H., DUCHAINEAU M., MA K.-L.: A Scalable, Hybrid Scheme for Volume Rendering Massive Data Sets. *Eurographics Symposium on Parallel Graphics and Visualization* (2006). 3

[CN93] CULLIP T. J., NEUMANN U.: *Accelerating Volume Reconstruction with 3D Texture Hardware*. Tech. rep., University of North Carolina at Chapel Hill, 1993. 2, 3

[DCH88] DREBIN R. A., CARPENTER L., HANRAHAN P.: Volume Rendering. *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1988). 2

[EBA12] EILEMANN S., BILGILI A., ABDELLAH M.: Parallel Rendering on Hybrid Multi-GPU Clusters. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012). 1

[FCS*10] FOGAL T., CHILDS H., SHANKAR S., KRÜGER J., BERGERON R. D., HATCHER P.: Large Data Visualization on Distributed Memory Multi-GPU Clusters. In *High Performance Graphics* (2010). 3

[GPC*15] GROSSET A. V. P., PRASAD M., CHRISTENSEN C., KNOLL A., HANSEN C. D.: TOD-Tree: Task-Overlapped Direct Send Tree Image Compositing for Hybrid MPI Parallelism. *Eurographics Symposium on Parallel Graphics and Visualization* (2015). 1, 3, 4, 6

[HBC12] HOWISON M., BETHEL E. W., CHILDS H.: Hybrid Parallelism for Volume Rendering on Large-, Multi-, and Many-Core Systems. *IEEE Transactions on Visualization Computer Graphics* (2012). 1, 2

[Hot18] HOTA A.: Building VisIt+OSPRay/SWR, 2018. URL: https://goo.gl/45uRwS. 3

[Hsu93] HSU M. W.: Segmented Ray-Casting for Data Parallel Volume Rendering. *IEEE Parallel Rendering Symposium* (1993). 1, 2, 3, 5

[KTW*11] KNOLL A., THELEN S., WALD I., HANSEN C. D., HAGEN H., PAPKA M. E.: Full-Resolution Interactive CPU Volume Rendering with Coherent BVH Traversal. In *IEEE Pacific Visualization Symposium* (2011). 2

[Kui91] KUIJK A. A.: *Advances in Computer Graphics Hardware III*. 1991. 2

[KVC*11] KUMAR S., VISHWANATH V., CARNS P., SUMMA B., SCORZELLI G., PASCUCCI V., ROSS R., CHEN J., KOLLA H., GROUT R.: PIDX: Efficient Parallel I/O for Multi-resolution Multi-dimensional Scientific Datasets. In *IEEE Conference on Cluster Computing* (2011). 2, 4

[KW03] KRÜGER J., WESTERMANN R.: Acceleration Techniques for GPU-based Volume Rendering. In *IEEE Visualization* (2003). 2

[KWN*13] KNOLL A., WALD I., NAVRÁTIL P. A., PAPKA M. E., GAITHER K. P.: Ray Tracing and Volume Rendering Large Molecular Data on Multi-core and Many-core Architectures. *International Workshop on Ultrascale Visualization* (2013). 1

[Lev88] LEVOY M.: Display of Surfaces from Volume Data. *IEEE Computer Graphics Applications* (1988). 2

[Lev90] LEVOY M.: Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics* (1990). 2

[MEFC94] MOLNAR S., ELLSWORTH D., FUCHS H., COX M.: A Sorting Classification of Parallel Rendering. *IEEE Computer Graphics and Applications* (1994). 2

[MKM99] MOSER R. D., KIM J., MANSOUR N. N.: Direct Numerical Simulation of Turbulent Channel Flow Up to $Re_\tau = 5200$. *Physics of Fluids* (1999). 1, 7

[MKPH11] MORELAND K., KENDALL W., PETERKA T., HUANG J.: An Image Compositing Solution at Scale. In *International Conference for High Performance Computing, Networking, Storage and Analysis* (2011). 2, 6

[MPHK93] MA K. L., PAINTER J. S., HANSEN C. D., KROGH M. F.: A Data Distributed, Parallel Algorithm for Ray-Traced Volume Rendering. In *IEEE Symposium on Parallel rendering* (1993). 2

[MRH10] MENSMANN J., ROPINSKI T., HINRICHS K. H.: An Advanced Volume Raycasting Technique using GPU Stream Processing. *International Conference on Computer Graphics Theory and Applications* (2010). 2

[PGH06] PARKER S., GUILKEY J., HARMAN T.: A Component-Based Parallel Infrastructure for the Simulation of Fluid–Structure Interaction. *Engineering with Computers 22*, 3-4 (2006). 2

[PGR*09] PETERKA T., GOODELL D., ROSS R., SHEN H.-W., THAKUR R.: A Configurable Algorithm for Parallel Image-Compositing Applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis* (2009). 2

[PM12] PHARR M., MARK W. R.: ISPC: A SPMD Compiler for High-Performance CPU Programming. In *Innovative Parallel Computing (InPar)* (2012). 2

[RWCB15] RATHKE B., WALD I., CHIU K., BROWNLEE C.: SIMD Parallel Ray Tracing of Homogeneous Polyhedral Grids. *Eurographics Symposium on Parallel Graphics and Visualization* (2015). 2

[Sab88] SABELLA P.: A Rendering Algorithm for Visualizing 3D Scalar Fields. *ACM SIGGRAPH Computer Graphics* (1988). 2

[SLM04] SCHROEDER W. J., LORENSEN B., MARTIN K.: *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*. 2004. 3, 5

[UAB*17] USHER W., AMSTUTZ J., BROWNLEE C., KNOLL A., WALD I.: Progressive CPU Volume Rendering with Sample Accumulation. In *Eurographics Symposium on Parallel Graphics and Visualization* (2017). 2

[VSW*17] VIERJAHN T., SCHNORR A., WEYERS B., DENKER D., WALD I., GARTH C., KUHLEN T. W., HENTSCHEL B.: Interactive Exploration of Dissipation Element Geometry. In *Eurographics Symposium on Parallel Graphics and Visualization* (2017). 2

[Wes90] WESTOVER L.: Footprint Evaluation for Volume Rendering. *ACM SIGGRAPH* (1990). 3

[WFM11] WHITLOCK B., FAVRE J. M., MEREDITH J. S.: Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Eurographics Conference on Parallel Graphics and Visualization* (2011). 9

[WJA*17] WALD I., JOHNSON G. P., AMSTUTZ J., BROWNLEE C., KNOLL A., JEFFERS J., GÜNTHER J., NÁVRATIL P.: OSPRay - A CPU Ray Tracing Framework for Scientific Visualization. *IEEE Transactions on Visualization Computer Graphics* (2017). 2

[WKJ*15] WALD I., KNOLL A., JOHNSON G. P., USHER W., PASCUCCI V., PAPKA M. E.: CPU Ray-Tracing Large Particle Data with Balanced P-K-D Trees. In *IEEE Scientific Visualization Conference* (2015). 2

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Transactions on Graphics* (2014). 2

[YCM08] YU H., CHAOLI WANG, MA K.-L.: Massively Parallel Volume Rendering using 2-3 Swap Image Compositing. In *IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis* (2008). 2