INVESTIGATING DEPTH OF FIELD IN VOLUME RENDERING AND DISTRIBUTED VOLUME RENDERING ON HIGH PERFORMANCE COMPUTING SYSTEMS

by

Pascal Grosset

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

August 2016

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Pascal Grosset**

has been approved by the following supervisory committee members:

Charles Hansen	, Chair	**5-5-2016** Date Approved
Mary Hall	, Member	**4-25-2015** Date Approved
Christopher Johnson	, Member	**4-25-2016** Date Approved
Valerio Pascucci	, Member	**4-25-2016** Date Approved
Georges-Pierre Bonneau	, Member	**4-25-2016** Date Approved

and by **Ross Whitaker**, Chair/Dean of

the Department/College/School of **Computing**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

The aim of direct volume rendering is to facilitate exploration and understanding of three-dimensional scalar fields referred to as volume datasets. Improving understanding is done by improving depth perception, whereas facilitating exploration is done by speeding up volume rendering. In this dissertation, improving both depth perception and rendering speed is considered. The impact of depth of field (DoF) on depth perception in direct volume rendering is evaluated by conducting a user study in which the test subjects had to choose which of two features, located at different depths, appeared to be in front in a volume-rendered image. Whereas DoF was expected to improve perception in all cases, the user study revealed that if used on the back feature, DoF reduced depth perception, whereas it produced a marked improvement when used on the front feature. We then worked on improving the speed of volume rendering on distributed memory machines. Distributed volume rendering has three stages: loading, rendering, and compositing. In this dissertation, the focus is on image compositing, more specifically, trying to optimize communication in image compositing algorithms. For that, we have developed the Task Overlapped Direct Send Tree image compositing algorithm, which works on both CPU- and GPU-accelerated supercomputers, which focuses on communication avoidance and overlapping communication with computation; the Dynamically Scheduled Region-Based image compositing algorithm that uses spatial and temporal awareness to efficiently schedule communication among compositing nodes, and a rendering and compositing pipeline that allows both image compositing and rendering to be done on GPUs of GPU-accelerated supercomputers. We tested these on CPU- and GPU-accelerated supercomputers and explain how these improvements allow us to obtain better performance than image compositing algorithms that focus on load-balancing and algorithms that have no spatial and temporal awareness of the rendering and compositing stages.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOTATION AND SYMBOLS

Central Processing Unit (CPU)

Swiss National Supercomputing Centre (CSCS)

X-ray Computed Tomography (CT)

Compute Unified Device Architecture (CUDA)

Depth of Field (DoF)

Directional Occlusion Shading (DOS)

Direct Volume Rendering (DVR)

Dynamically Scheduled Region-Based (DSRB)

OpenGL Shading Language (GLSL)

General Purpose computing on GPU (GPGPU)

Graphics Processing Unit (GPU)

High Performance Computing (HPC)

Institute of Electrical and Electronics Engineers (IEEE)

Image Composition Engine for Tiles (IceT)

Message Passing Interface (MPI)

Magnetic Resonance Imaging (MRI)

National Energy Research Scientific Computing Center (NERSC)

Open Graphics Library (OpenGL)

Open Multi-Processing (OpenMP)

Remote Direct Memory Access (RDMA)

Single Instruction Multiple Data (SIMD)

Texas Advanced Computing Center (TACC)

Task-Overlapped Direct Send Tree Image Compositing (TOD-Tree)

Three-Dimensional (3D)

ACKNOWLEDGMENTS

My principal thanks goes to my adviser, Professor Charles (Chuck) Hansen, for his patience and guidance and for finding funding and supercomputing time to allow me to do the many scaling studies that were essential for my Ph.D.

I would also like to thank the members of my Ph.D. committee: Mary Hall, for mentoring me on my first publication; Chris Johnson, for providing the nicest possible environment for graduate students at the SCI Institute; Valerio Pascucci, from whom I learned about topology; and Georges-Pierre Bonneau, from whom I learned about depth perception during the three months I spent at INRIA.

Many other people have supported me indirectly in this endeavor. The list is long but some deserve a special mention here: Aaron Knoll, who has always been keen to discuss ideas and review my publications; Bill Thompson, whose class taught me how to make presentations; Karen Feinauer and Ann Carlstrom, for providing me with the documentation I constantly needed for my J-1 visa; Begum Durgahee and Anwar Chutoo, who have always been around when the going got tough; Manasa Prasad, who has been so much fun to hang around with and still is; Avinash Meetoo, who was an unofficial mentor in my first job and introduced me to much of the Linux I know. Lastly, thanks to my parents who, though far away, have always been supportive.

Finally, my research was made possible by the Fulbright program and the U.S. Department of State for funding me during the first three years of my Ph.D, and the following grants: DOE NNSA Award DE-NA0000740, KUS-C1-016-04 made by King Abdullah University of Science and Technology (KAUST), Award DE-NA0002375: (PSAAP) Carbon-Capture Multidisciplinary Simulation Center, DOE SciDAC Institute of Scalable Data Management Analysis and Visualization DOE DE-SC0007446, NSF OCI-0906379, NSF IIS-1162013, NIH-1R01GM098151-01, and NSF ACI-1339881.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Volume rendering for visualization has become increasingly popular among researchers and scientists since its introduction in the mid-1980s [2] [3]. It can be used to visualize volumetric data acquired through simulations, such as combustion and climate simulation, or through a scanning process, such as Magnetic resonance imaging (MRI) and X-ray computed tomography (CT) scans. Volume rendering is now available in many commonly used visualization packages such as Paraview [4] and VisIt [5].

Volume data is usually visualized using either direct or indirect volume rendering. Approaches that generate an intermediate representation and use the intermediate representation to create an image are referred to as indirect volume rendering. An example is the Marching Cubes [6] algorithm, which generates a polygonal representation of an isosurface and is then rendered to an image. In direct volume rendering, volume data is directly used to generate an image without any intermediate stage. For example, in ray casting direct volume rendering, an image is created by sending a ray through each pixel of the image. Data is sampled at regular intervals along the ray and each sample is mapped to a color and opacity using a transfer function. The colors are then blended to produce one color for each pixel. In this dissertation, the focus is on direct volume rendering (DVR).

The goal of volume rendering is to allow scientists to explore and understand datasets. In practice, this is achieved by improving the quality and speed of volume rendering. Increasing the quality, by adding techniques such as global illumination and shadows, makes it easier for scientists to see the features present in a dataset, thereby improving understanding. Increasing the rendering speed enables better interaction with a dataset, thereby improving exploration of the dataset. In this

dissertation, improving both the quality and speed of rendering is considered. More precisely, a user study was carried out to determine how depth of field can be used to improve depth perception of volume-rendered images, and new image compositing techniques have been developed for distributed volume rendering to improve rendering speeds.

1.2 Evaluation of Depth of Field for Depth Perception in Direct Volume Rendering

The human visual system is able to perceive depth because of a number of depth cues. The more depth cues available to us, the easier it is for us to understand the depth relation of objects in an image. However, volume-rendered images usually have a limited set of depth cues. Very often, the only depth cues present in a volume-rendered image are interposition, close objects overlap far away objects; perspective, far away objects appear closer than when close to us; and occasionally shadows, objects are in front of the shadows they cast.

For solid surfaces, shadows are useful in helping us estimate the relative ordering of features. Lindemann et al. [7] conducted a user study on the use of different illumination methods in DVR and found that directional occlusion shading [8] brings about 20% improvement in ordinal depth perception. However, for highly translucent surfaces, directional occlusion shading is less effective as it will only darken a translucent image. As pointed out by Boucheny et al. [9], estimating depth in a highly transparent scene can be very difficult, and DVR images often have highly transparent features. In Fig. 1.1, the "static" subtree shows the depth cues available in a volume-rendered image. Shadow and interposition will not help for very translucent surfaces. Aerial, texture, and familiar size apply to objects that are far away, which is not applicable in a volume-rendered image. Perspective is usually already available. Therefore, we will try to add focusing by using depth of field, which will make objects not in focus appear blurry. Since depth of field (DoF) is a quite recent addition to direct volume rendering, its impact in volume-rendered images has not been extensively studied. The only study available looked at the impact of depth of field in angiography images [10]. In

Chapter 3 of this dissertation, a user study on the impact of DoF, implemented using the technique proposed by Schott et al. [1], is described along with a discussion on how to best benefit from depth of field effects in direct volume rendering.

1.3 Image Compositing for OpenMP/Hybrid MPI Parallelism on CPU-Enhanced Supercomputers

With the increasing availability of supercomputers, scientists are running increasingly larger simulations that generate huge amounts of data. Since it is impractical to move the results of these simulations, DVR is done directly on these supercomputers. Most supercomputers are distributed memory systems that have thousands of nodes, each of which has one or more powerful processing units with several gigabytes of memory, connected through a high-speed network. When rendering on these systems, the workload is typically divided so that each node gets an equal amount of work. In sort-last parallel rendering, the most common approach for DVR in distributed memory systems, each node always loads a section of the data that it renders to an image. The images are then exchanged and blended in the compositing stage to create the final image. When rendering on many nodes, image compositing can become the bottleneck due to the number of image exchanges that are required.

Image compositing has been studied for a number of years, and many algorithms such as direct send [11], binary swap [12], and radix-k [13] have been developed to handle image compositing on distributed memory systems. However, the increase in computing power of supercomputers in recent years has not been matched by a similar increase in communication speed, making computation cheap compared to communication. Yet, algorithms such as binary swap and radix-k tend to focus on equally dividing the computation workload rather than trying to find ways of minimizing communication.

In Chapter 4, a new image compositing algorithm is presented that tries to minimize communication and overlaps communication with computation to try to hide communication latencies. Also, since Howison et al. [14] showed that using one MPI rank per node is more efficient than using one MPI rank per core,

our algorithm uses one MPI rank per node with threads and auto-vectorization to make full use of the cores and SIMD parallelism of modern multicore CPUs. We tested our algorithm on the Stampede supercomputer at TACC and the Edison supercomputer at NERSC to show that our algorithm is, most of the time, faster than binary swap and radix-k from the ICET library [15].

1.4 Distributed Volume Rendering Pipeline With Image Compositing on GPU-Enhanced Supercomputers

GPUs have been successfully used on desktop computers for DVR. Ray casting for volume rendering on GPUs has been reported to be at least 1.5X faster than on CPUs [16], and volume rendering packages, such as ImageVis3D [17] [18], can interactively render medium-sized datasets on consumer-grade desktop GPUs. Supercomputers, such as Titan at Oak Ridge National Lab and Piz Daint at the Swiss National Supercomputing Center, are both enhanced with GPUs. When doing volume rendering on these systems, using the GPUs on these supercomputers instead of the CPU is the best choice to guarantee fast rendering.

Until recently, communication between GPUs found on different nodes of a supercomputer was very costly. Since image compositing in sort-last parallel rendering is a communication-intensive task, image compositing was often handled by the CPU instead of the GPU to avoid the high cost of inter-node GPU communication. Volume-rendered images generated on the GPU were copied to the CPU for image compositing, which is inefficient since the GPU to CPU copy operation is expensive, and GPUs are faster at blending images than CPUs. With the introduction of GPU Direct RDMA, inter-node GPU communication now requires only one copy operation instead of five copy operations. There is, then, no need to copy data to the CPU prior for image compositing anymore.

In Chapter 5, a distributed volume rendering pipeline using OpenGL shaders for raycasting rendering, and CUDA kernels with GPU Direct RDMA for compositing is presented. We compared the image compositing speed on the Piz Daint GPU-enhanced supercomputer against the Edison CPU-enhanced supercomputer at NERSC and show that we get comparable or even better speed on Piz Daint,

showing that the full distributed volume rendering pipeline can now be implemented on the GPU.

1.5 Dynamically Scheduled Region-Based Image Compositing

One of the common assumptions of image compositing algorithms in distributed volume rendering is that all the nodes will finish rendering and start compositing at the same time. However, this is very rarely the case unless we have several thousands nodes for rendering. There are many reasons for this: firstly, it is rare for datasets to have a uniform distribution of data. Secondly, when using perspective projection, nodes closer to the camera produce a larger image compared to nodes far from the camera. Rendering a larger image takes more time than rendering a smaller image. Finally, if the user zooms in on one specific region of a dataset, part of the dataset might fall outside the viewing window and not need to be rendered at all.

Moreover, this difference in rendering speed is further increased if lighting is used and normals need to be calculated. Furthermore, many visualization clusters are medium-sized distributed memory machines where there are hundreds rather than thousands of nodes. The time taken to render a large image can be substantially greater than the time to render a small image on these clusters. If we do not want the uneven rendering to slow down compositing, nodes that are done rendering should exchange images only with nodes that are done rendering, and not wait for nodes that are still rendering.

In Chapter 6 of this dissertation, a new image compositing algorithm with spatial and temporal awareness is presented. We test the algorithm, using a combustion and two artificial test datasets, on the Edison supercomputer at NERSC for up to 2,048 nodes and show that it is faster than traditional image compositing algorithms such as radix-k and TOD-Tree [19].

1.6 Thesis Statement

In this dissertation, we show that depth of field can be used to improve depth perception in volume-rendered images, and that image compositing algorithms

that focus on communication (minimizing communication, overlapping communication with computation, and correctly scheduling communication) perform better than algorithms that focus on balancing the workload.

1.7 Dissertation Contributions

The main contributions of this dissertation are:

- **A user study on the impact of depth of field in DVR** is conducted and a set of guidelines on how to use depth of field effects to improve ordinal depth perception in DVR is presented. We show that depth of field effects can improve correct ordinal depth perception by up to 20% if used correctly but can worsen ordinal depth perception if used improperly. This work has been published in the IEEE Pacific Visualization Symposium Symposium, 2013 [20].
- Evaluation of depth of field for depth perception in DVR [20], IEEE Pacific Visualization Symposium Symposium 2013, explored how depth of field can be used to improve depth perception in volume rendering.
- **TOD-Tree: Task-Overlapped Direct send Tree Image Compositing for Hybrid MPI Parallelism**. This is a new image compositing algorithm for hybrid OpenMP/MPI parallelism that focuses on minimizing communication latencies and is generally faster image compositing algorithms such as binary swap and radix-k. We tested this algorithm on an artificial dataset and a combustion dataset for up to 4,096 nodes of the Edison supercomputer. This work has been published in the Eurographics Symposium on Parallel Graphics and Visualization, 2015 [19].
- **A very low latency pipeline for doing ray casting volume rendering and image compositing on GPU-enhanced supercomputers**. We also use GPU Direct RDMA and CUDA kernels to create an image compositing algorithm that runs exclusively on GPUs and matches the compositing speeds of image compositing algorithms on CPUs. The algorithm was tested on up to 4096 nodes of the Piz Daint GPU-enhanced supercomputer, and we show that the performance matches image compositing on CPU-enhanced supercomputers.

This work has been accepted to the IEEE Transactions on Visualization and Computer Graphics, 2016 [21].

- **An image compositing algorithm with spatial and temporal awareness** to guide the exchange of images in distributed volume rendering and avoid waiting on nodes that are still rendering. The algorithm was tested on the Edison supercomputer at NERSC for up to 2048 nodes, and we show that we achieve better performance than traditional compositing algorithms such as radix-k and TOD-Tree that have no spatial and temporal knowledge of rendering and compositing. This work has been accepted to the Eurographics Symposium on Parallel Graphics and Visualization, 2016 [22].

1.8 Outline

The remainder of this dissertation is organized as follows. The background and important related work are explained in Chapter 2. Chapter 3 details the user study that was carried out on the use of depth of field in direct volume rendering. Chapter 4 presents the image compositing algorithm for hybrid OpenMP/MPI parallelism on CPU-enhanced supercomputers. In Chapter 5, the compositing algorithm is extended to a GPU-enhanced supercomputer, and a pipeline for distributed volume rendering using GPUs for both rendering and compositing is presented. In Chapter 6, an image compositing algorithm with spatial and temporal awareness is presented that outperforms traditional image compositing algorithms. Finally, conclusions and future work are given in Chapter 7.

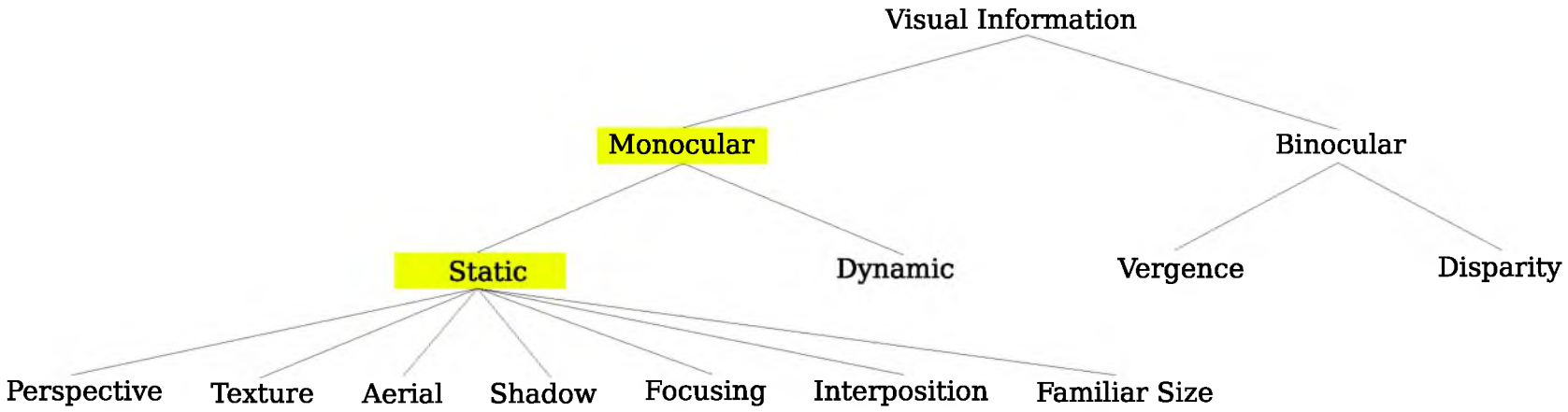

Fig. 1.1. Categorization of depth cues.

CHAPTER 2

RELATED WORK

2.1 Direct Volume Rendering

Direct volume rendering is a technique that generates an image from blocks of scalar data without having to create any intermediate representation. There are three main approaches to DVR: splatting, texture-based methods, and ray casting. In the splatting approach, Fig. 2.1, each voxel is represented by a Gaussian kernel that is scaled by the scalar value at the voxel. The volume is drawn from the back to the front where each of these kernels is projected onto the screen. The image produced is usually blurry and not very appealing. In the texture-based approach, Fig. 2.2, the volume is converted to a 3D texture that is stored in the memory of a GPU. Polygonal slices parallel to the view-plane are used to sample the volume and are blended in a front-to-back or back-to-front manner. The DVR approach used for depth of field in Chapter 3 uses this texture-based approach. Finally, in the ray casting approach, Fig. 2.3, for each pixel in an image, a ray is cast in the 3D scalar data. For each ray, scalar values inside the volume are sampled at regular intervals along the ray, and a color is assigned to each sampled scalar value using a transfer function. The colors along a ray are then blended to get the final color of that pixel. When rendered on one computer, there are two stages, namely, loading the data from disk and creating an image from the data using one of the direct volume rendering techniques mentioned above. We are first going to look at applying depth of field to DVR.

2.2 Depth Perception Cues

To be able to perceive the depth of objects in the real world and in a synthetic image, the human visual system makes use of a number of depth cues. Depth cues have been studied thoroughly [23], [24] and are generally grouped into two

categories: monocular and binocular. Binocular refers to the interocular offset to perceive depth in a scene, and monocular depth cues refers to having a single image of the scene. In a typical volume-rendered image, only one view is being shown on the screen and there is no motion. Therefore, the only depth cues available are static monocular depth cues, which are described in Table 2.1.

Depth of focus is one of the depth cues the human visual system uses to perceive depth. The shape of the lens in a human eye is distorted by ciliary muscles to focus light on the retina. A schematic representation of the human eye is shown in Fig. 2.4. The process of modifying the shape of the lens of our eye is called accommodation. The human visual system can use information about the amount of distortion of the lens from the ciliary muscles as a depth cue; especially for very close objects [24]. Also, the part of the image not in focus, in front and behind the focal plane, will appear blurred. Using the amount of blurriness of an image as a depth cue is not a viable indication of how close or far an object is. As indicated by Mather et al. [25], blur discrimination is poor in humans.

2.3 Depth of Field in Computer Graphics

Several attempts have been made to reproduce DoF in computer graphics. Barsky et al. [26] did a comprehensive study of different DoF techniques in computer graphics.

Many of these approaches are image space techniques that blur an initially generated in-focus image. Potmesil and Chakravarty [27] use linear filtering in a postprocessing stage to adaptively blur images according to their distance from the focal plane. This technique is fast, since it does not attempt to minimize depth discontinuities or color bleeding. Distributed ray tracing has also been used to compute depth of field effects. One of the earliest techniques, from Cook et al. [28], uses multiple rays to simulate ray tracing. However, interactive approximations for computing depth of field effects are rarely based on ray tracing due to the high performance cost.

In interactive computer graphics, a common technique is to sort the scene by depth and apply different amounts of blur to different levels of depth. This

technique was used by Barsky et al. [29] and Kraus et al. [30]. In DVR, DoF has been proposed by Crassin et al. [31] for large volumes in the Gigavoxels Framework. Ropinski et al. [10] use DoF for angiography images and they also conduct a user study on the use of modified DoF to enhance spatial understanding. In their work, they try to determine where the user is focusing in an image. The part deemed to be behind the region in focus has DoF effects applied to it. Consequently, there are no DoF effects in front of the focal plane. From their user study, they find that DoF helps to improve the correctness of depth perception, but they also saw an increase in response time. They attribute this to the user having to get used to some part of the image appearing out of focus.

2.4 Depth of Field Implementation

In this dissertation, DoF was implemented as shown in Fig. 2.5 and described by Schott et al. [1]. A GPU slice-based volume renderer is used, and the scene is broken down into two regions, namely, before and after the focal plane. For the region between the focal plane and the front of the volume (where the camera is), the volume is processed from the front of the volume to the focal plane (in a front-to-back manner), and each slice is blurred according to its position. For the part behind the focal plane, the volume is processed from the end of the volume to the focal plane (in a back-to-front manner). Each rendered slice is blended with a blurred region of the previous slice. The blur kernel's size decreases as the slice approaches the focal plane. Two directions are needed to ensure that the in-focus region does not contribute to the blurred region. A more detailed description of the algorithm with pseudocode can be found in Schott et al. [1].

2.5 Perceptive Studies for Depth of Field

Depth of field is simulated using blur in computer graphics. The usefulness of blur has been debated among researchers for some time. Whereas Held et al. [32] indicated that blur is a stronger depth cue than previously thought, Mather et al. [33] found that blur on its own gives only about 75% correct depth results. However, combined with other depth cues, the usefulness of blur is increased to about 95%. This was confirmed by Held et al. [34], who concluded that though blur on its own

did not help to estimate absolute or relative distance, combined with perspective projection, it becomes a quite useful depth cue. In volume-rendered images, we will usually have other depth cues available apart from blur. Perspective projection is usually cheap to add and occlusion is usually available except for translucent images. So, since depth of field will not be used on its own in volume rendering, it should be a useful depth cue.

2.6 Distributed Volume Rendering

Distributed volume rendering is now commonly used in the scientific visualization community. There are three approaches [35] to parallel rendering on distributed memory systems: sort-first, sort-middle, and sort-last, shown in Fig. 2.6. In sort-first, the data is partitioned based on the viewpoint, and each node then loads and renders a section of the final image. In sort-middle, each node always loads the same data, but after sampling is done, the data is redistributed based on the viewpoint. Finally, in sort-last, each node loads a section of the dataset that it renders to an image. These images are then blended together during the compositing stage to produce a full rendered image of the dataset. No communication is required in the loading and rendering stages, but the compositing stage can require extensive communication. When doing sort-last distributed volume rendering on few nodes of a distributed memory machine, rendering is usually slower than the compositing. However, as the number of nodes increases, each node has less and less data to render but more communication is required. At some point, then, compositing becomes more expensive than rendering and fast image compositing algorithms become essential for sort-last image compositing.

2.7 Image Compositing Algorithms

Since sort-last image compositing is the most commonly used approach for distributed volume rendering, several algorithms have been developed to speed up image compositing. One of the oldest compositing algorithm is direct send. Direct send can refer to serial or parallel direct send. In serial direct send, all the processes send their data directly to the display process, which blends the images. In parallel direct send [11], [36], shown in Fig. 2.7, each process takes

responsibility for one section of the final image, gathers data for that section from all the other processes, and blends these sections. Then, during the compositing stage, the display process gathers the final section from all nodes. On GPUs, parallel direct send is popular because of its flexibility. Eilemann et al. [37] show that the performance of parallel direct send is comparable to binary swap and sometimes even better. Rizzi et al. [38] compare the performance of serial and parallel direct send, for which they get very good results as GPUs are very fast. However, in both cases, the main bottleneck is the network performance, which negatively affects the performance of the algorithm.

In binary tree compositing techniques [39], shown in Fig. 2.8, one of the leaves sends its data to the other leaf in the pair, which does the compositing. The leaf that has sent its data is now idle. The main issue with this technique is half of the nodes go idle at each stage, which results in load imbalances. However, now that computation is reasonably cheap, this could again be a viable technique, but tree compositing techniques also send full images at each stage, making communication slow. Binary-swap by Ma et al. [35], shown in Fig. 2.9, improves the load balancing of binary tree compositing by keeping all the processes active in compositing until the end. The processes are grouped in pairs, and initially, each process in the pair takes responsibility for half of the image. Each process sends the half it does not own and blends the half it owns. In the next stage, processes that are authoritative on the same half exchange information in pairs again, so that each is now responsible for a quarter of the image. Compositing proceeds in stages until each process has 1/p of the whole image where p is the number of process involved in the compositing. Once this is done, each process sends its section to the display process. Binary-swap has been subsequently extended by Yu et al. [40] to deal with non-power of two processes.

In Radix-k, introduced by Peterka et al. [13], shown in Fig. 2.10, the number of processes p is factored in r factors so that k is a vector where $k = [k_1, k_2, ..., k_r]$. The processes are arranged into groups of size k_i and exchange information using direct send. At the end of a round, each process is authoritative on one section of the image in its group. In the next round, all the processes with the same authoritative

partition are arranged in groups of size k_{i+1} and exchange information. This goes on for r rounds until each process is the only one authoritative on one section of the image. Both binary-swap and radix-k have a gather stage in which the display process has to gather the data spread among the p processes. If the vector k has only one value that is equal to p, radix-k behaves like direct send. If each value of k is equal to 2, then it behaves likes binary-swap. Radix-k, binary-swap, and direct send are all available in the IceT package [15], which also adds several optimizations [41].

One of the ways of reducing the number of messages needed for image compositing is to do volume rendering using one MPI rank per node instead of one MPI rank per core. Howison et al. [42] [14] compared distributed volume rendering using only MPI versus using MPI and threads and found that using MPI and threads minimized the exchange of messages between nodes and resulted in faster volume rendering. However, for compositing, they only used MPI_Alltoallv, where processes exchange fragments using MPI direct send, but they did mention in their future work the need for better compositing algorithms.

2.8 Image Compositing With Spatial Awareness

However, although radix-k and binary swap are fast, they do no take into account the contents of the image from each rendering process. They all decide statically for which region a computing process should be responsible and stick to that allocation. A process, then, may be responsible for a region for which it does not have any initial content, which needlessly increases communication. However, some algorithms take into account the image contents of a node. The Scheduled Linear Image Compositing (SLIC) algorithm of Stompel et al. [43] ensures that the region to which a process is assigned is one to which it contributes. The contribution to the final image from each process is computed based on the data extents loaded by a process and the camera position. Scan lines of the overlapping regions are assigned to processes contributing to them in an interleaving fashion. Also, image regions that do not overlap with other images are directly sent to the display node

without any blending. Strengert et al. [44] used the SLIC algorithm for image compositing on GPU clusters.

Although SLIC has spatial awareness of the contribution of each rendering process, it does not have any temporal awareness, that is, it does not know when a process will finish rendering and is ready to participate in compositing. Moloney et al. [45] used an estimate on the cost to render a pixel to do dynamic load balancing using a sort-first rendering approach, and Muller et al. [46] used the previous rendering time in a time-varying dataset to predict the cost of rendering the current timestep. For this algorithm, we do not try to move the data around and estimate the rendering time. Instead, we communicate with the rendering nodes and schedule compositing accordingly.

2.9 Image Compositing on Specific Hardware

Also, recognizing that communication is the main bottleneck in image compositing, Pugmire at al. [47] used a Network Processing Unit (NPU) to speed up the communication and Cavin et al. [48] used shift permutation to get the maximum cross bisectional bandwidth from InfiniBand Fat-Trees to speed up communication. These improvements tie compositing algorithms to specific hardware network infrastructure, rather than providing a more general software solution.

2.10 Rendering and Compositing on the GPU

Many systems, such as Chromium [49] and Equalizer [50], have been developed for parallel rendering on GPUs. Direct volume rendering using either a slicing [51] or raycasting [52] approach has been done on the GPU. Muller et al. [53] and Fogal et al. [54] have developed distributed memory volume renderers for GPU that use shaders and OpenGL. For compositing, Fogal et al. used a tree-based compositing from IceT and Muller et al. used direct send. In both cases, compositing involved copying data out of the GPU before inter-node communication with MPI. Recently, Xie et al. [55] used up to 1024 GPUs for rendering on the Titan supercomputer, a Cray XK7 system, at Oak Ridge National Laboratory, but they used the CPU for image compositing. The only instance we found where it was explicitly mentioned that compositing was done on GPUs is the vl3 system by Rizzi et al. [38]. They

compared the performance of serial and parallel direct send scaling up to 128 Nvidia Tesla M2070 GPUs, but do not mention the use of GPU Direct RDMA for image compositing.

Currently, the only way for GPUs to communicate directly across a network is through CUDA. In 2011, Wang et al. [56] proposed an MPI design that integrates CUDA data movement with MPI; they achieved a 45% improvement in one-way latency. GPU Direct RDMA was then introduced in CUDA 5.0. Potluri et al. [57] mentioned an improvement in the latency by 69% and 32% for 4 Byte and 128 KB messages, respectively, for MPI_Send/MPI_Recv using GPU Direct RDMA on InfiniBand systems. Now, GPU Direct RDMA is available in MVAPICH2, OpenMPI, and CRAYMPI. In the worst case, without GPU Direct RDMA, five copies are needed, as shown in Fig. 2.11, to transfer data between GPUs found in different nodes. The data is first copied from the GPU's memory to the CUDA driver buffer's memory found in the main memory. Another copy transfers the data to the network driver buffer, also in main memory. The next copy takes the data across the network to the network driver buffer in the destination node. There, another copy is needed to transfer the data to the CUDA driver buffer and, finally, a last copy sends the data to the GPU's memory [58]. However, using GPU Direct RDMA, only one copy is required to transfer data between GPUs across nodes.

Since rendering is mostly done in OpenGL rather than CUDA, the CUDA OpenGL interoperability, provided as part of the CUDA Runtime API, can be used as a bridge between CUDA and OpenGL. Initially, it was not possible on Tesla class Nvidia GPUs used in supercomputers to run both CUDA and OpenGL at the same time, but this capability is now available in the Nvidia K20m, K20X, K40, and K80 GPUs [59]. The only additional requirement for running OpenGL is to have an X Server, which is needed to create an OpenGL context. Klein and Stone [60] describe how to get OpenGL working on a Cray XK7 accelerator. Also, some GPU-accelerated supercomputers, such as the Piz Daint supercomputer in Switzerland, have an X Server module that can be loaded as needed.

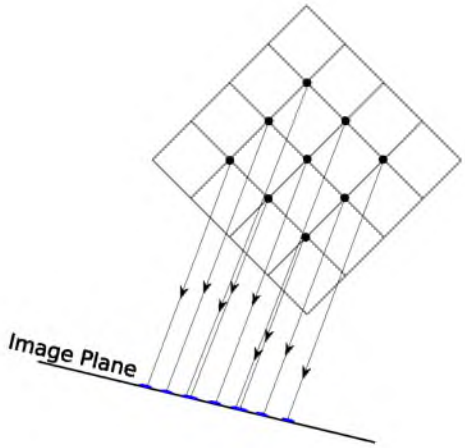

Fig. 2.1. Splatting for DVR. The arrows show the direction each sample is splatted onto an image plane.

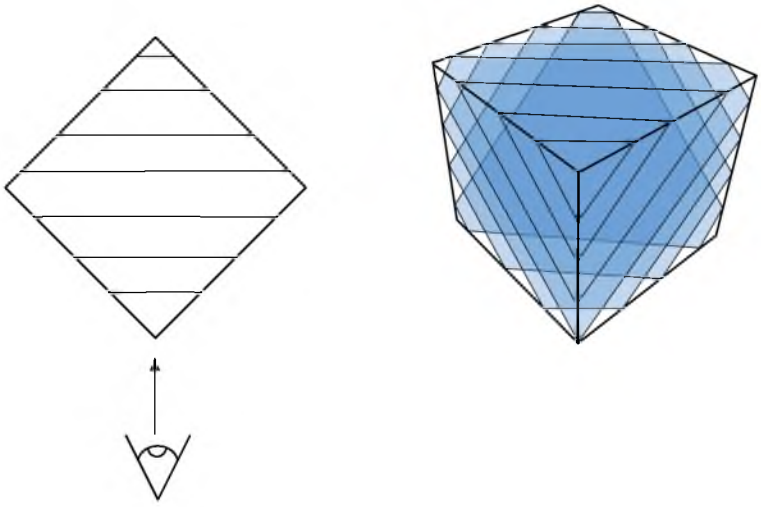

Fig. 2.2. Texture-based volume rendering. The top view is shown on the left and the camera aligned slices are shown on the right.

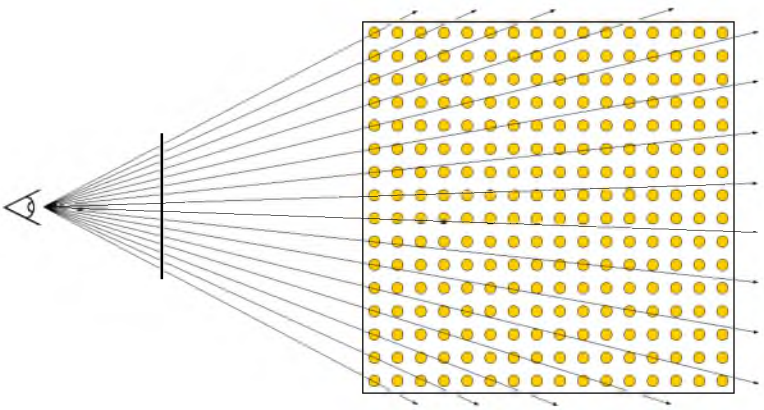

Fig. 2.3. Rays are sent through each pixel of the image plane and blended in a back-to-front or front-to-back manner.

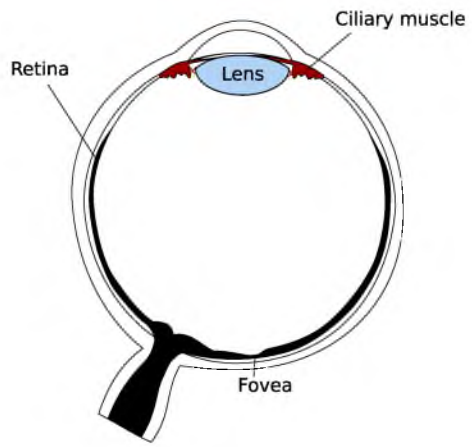

Fig. 2.4. Schematic representation of the human eye.

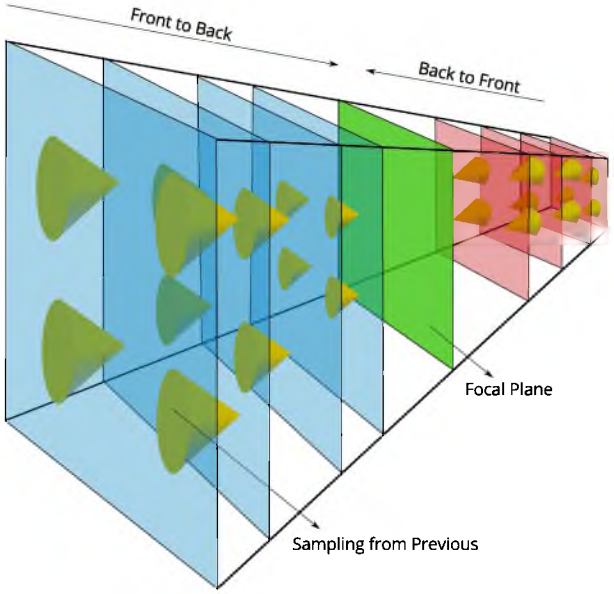

Fig. 2.5. Geometric setup of the DVR implementation of DoF.

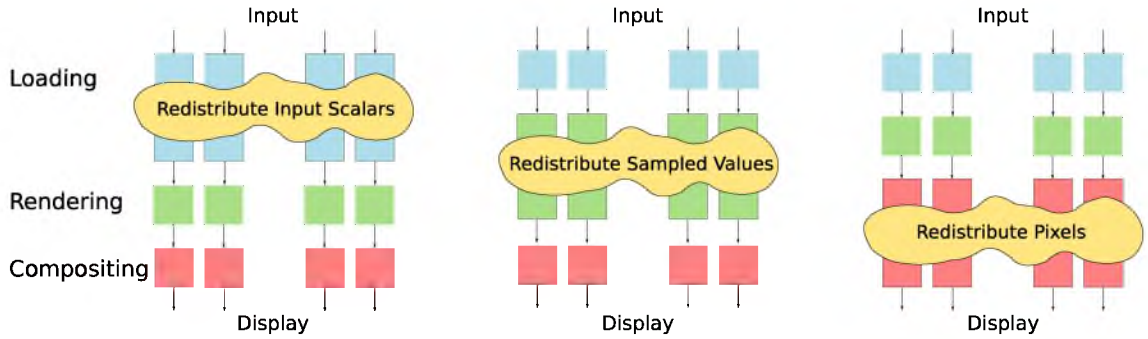

Fig. 2.6. Types of parallel rendering: left diagram is sort-first; middle diagram is sort-middle, and right diagram is sort-last.

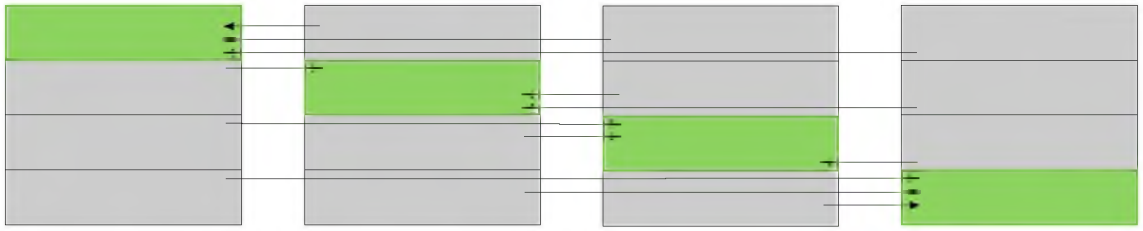

Fig. 2.7. Parallel direct send: the green rectangle shows the region each process is authoritative on. The arrows show the direction of the image exchanges. The gray rectangle indicates the region for which the process has data that it is not authoritative on, and will be sending out.

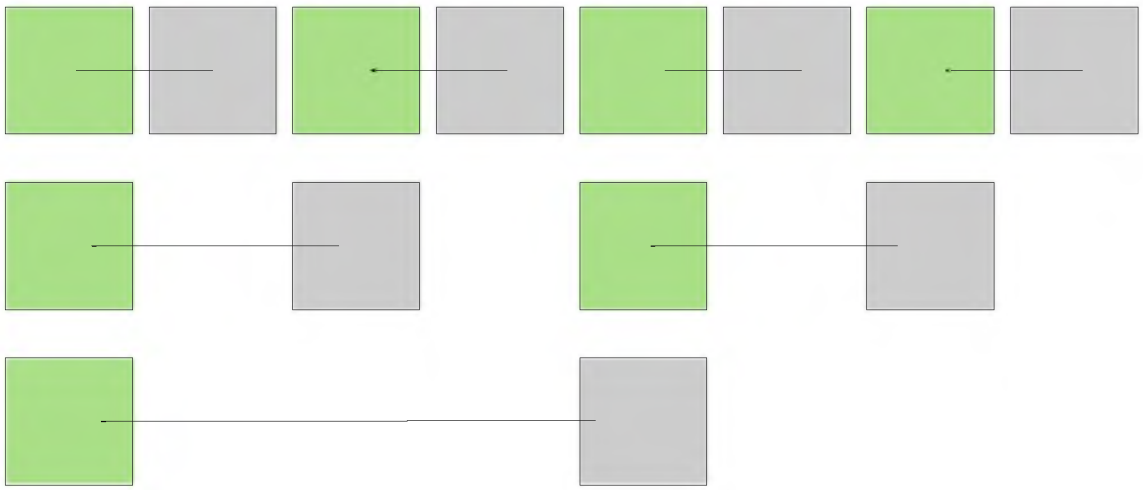

Fig. 2.8. Tree compositing: the green rectangle shows the region each process is authoritative on. The arrows show the direction of the image exchanges. The gray rectangle indicates the region for which the process has data that it is not authoritative on, and will be sending out.

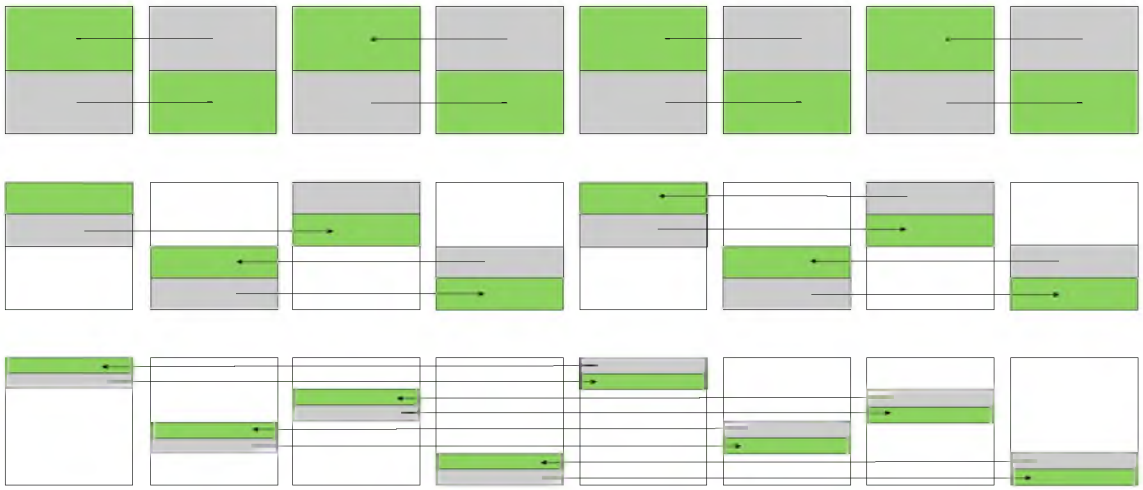

Fig. 2.9. Binary swap: the green rectangle shows the region each process is authoritative on. The arrows show the direction of the image exchanges. The gray rectangle indicates the region for which the process has data that it is not authoritative on, and will be sending out, and the white sections indicates regions for which a process has no data.

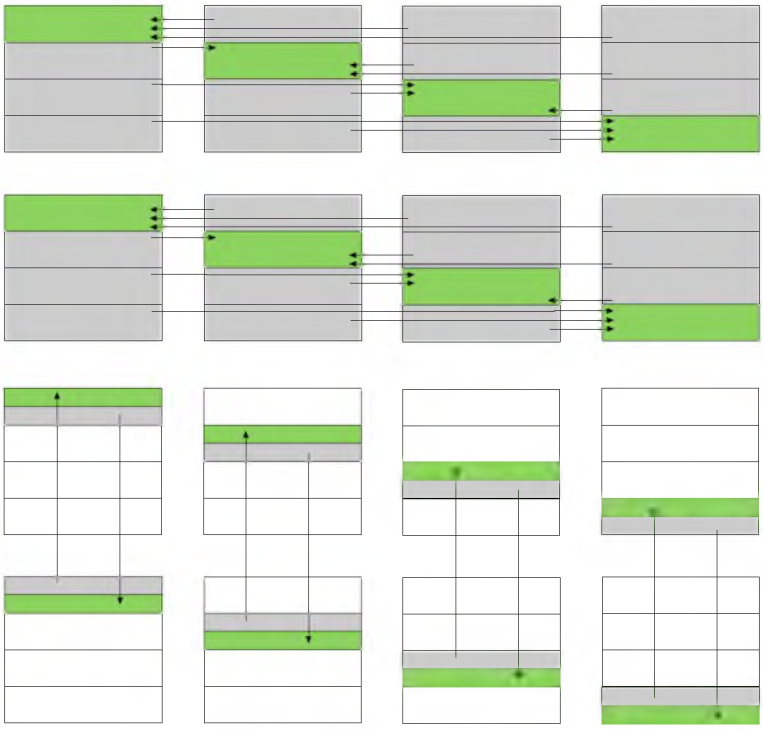

Fig. 2.10. Radix-k: the green rectangle shows the region each process is authoritative on. The arrows show the direction of the image exchanges. The gray rectangle indicates the region for which the process has data that it is not authoritative on, and will be sending out, and the white sections indicates regions for which a process has no data. Vector k in this case is 4 and 2.

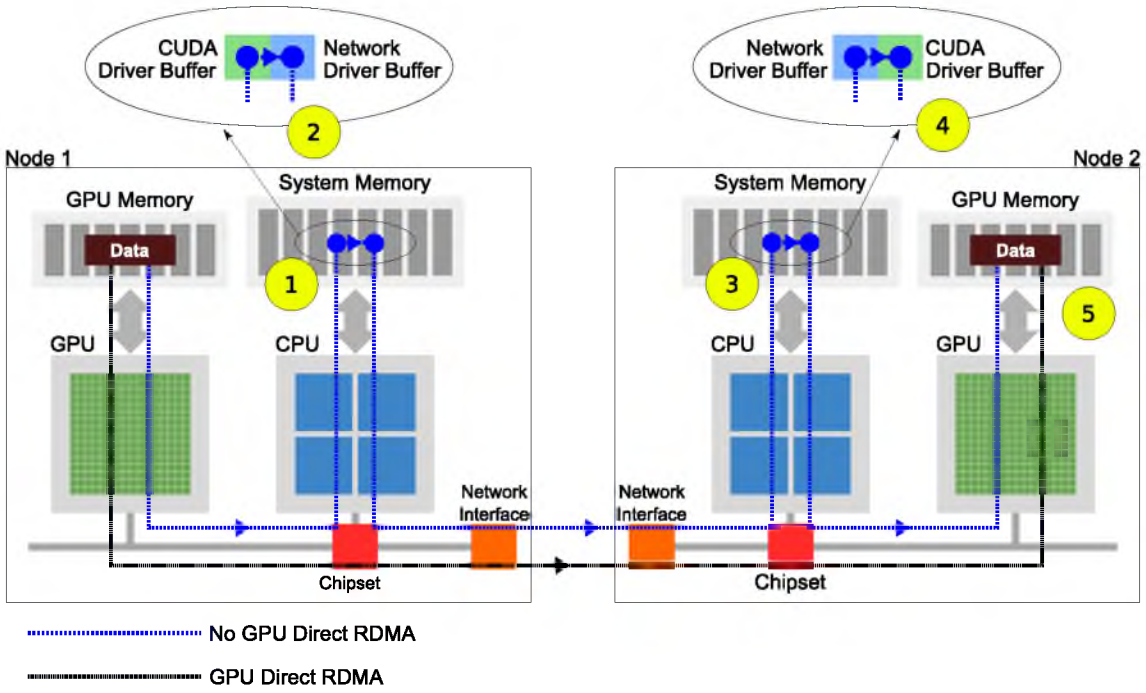

Fig. 2.11. Inter-node GPU communication with and without GPU Direct RDMA. The yellow circles show the copies: 1, copy from the GPU memory to the CPU memory in node 1; 2, copy from the CUDA Driver buffer to the network dirver buffer in the system memory of node 1; 3, copy across the interconnect from node 1's network driver buffer to node 2's network driver buffer; 4, copy from the network driver buffer to the CUDA driver buffer in node 2; and finally 5, copy from CUDA driver buffer to the GPU memory of the GPU in node 2

TABLE 2.1. Monocular static depth cues.

Depth Cue	Description
Atmospheric Perspective	Objects far away from us appear blurred and with a tint of blue.
Depth of Focus	Objects in focus appear sharp and those not in focus appear blurred.
Familiar Size	When we use our prior knowledge of the world to judge distances, e.g, the smaller a plane looks in the sky, the further away from us we know it is.
Occlusion	Objects in front of others overlap those at the back and so hide part of the back objects.
Perspective	Parallel lines appear to converge at infinity.
Relative Size	Objects that are far away from us take a smaller area in our field of view.
Shading Brightness	Objects that are far from us tend to appear more dimly lit than objects that are close to us.
Shadow	When we know where the light source is, we make use of where the shadow will fall to decide where the object is.
Texture Gradient	Fine details are clearly seen for objects that are close to us compared to objects that are far away.

CHAPTER 3

EVALUATING DEPTH OF FIELD FOR DEPTH PERCEPTION IN DVR

3.1 Introduction

Estimating the horizontal and vertical positions of features in a 2D image is quite easy, but estimating the depth of features is harder. Depth understanding relies on using depth cues that help us understand the position of features relative to each other in an image. In DVR, the common depth cues are perspective, occlusion, and shadows. Occlusion and shadows are particularly useful for solid surfaces. Lindemann et al. [7] conducted a user study on the use of different illumination methods [61] and found that directional occlusion shading [8] (DOS) brings about 20% improvement in ordinal depth perception. However, for highly translucent surfaces, as shown in Fig. 3.1, DOS is much less effective. In this chapter, we investigate the use of depth of field in DVR. Depth of field adds a focusing depth, in contrast to illustration methods, which add a shading/shadow depth cue. Focusing depth cues are less likely to be negatively affected by transparency.

In very simple terms, depth of field is the region of an image that appears to be sharp. For example, in Fig. 3.2, the two padlocks that are not blurred are in the depth of field region of the image; the padlocks before and after that region appear blurred. We also need to differentiate among the three kinds of depth descriptors: absolute, relative, and ordinal. Absolute descriptions are usually quantitative and are defined in units such as meters or relative to the viewer's body. Relative descriptors relate one property to another. They can be further broken down into relative and ordinal. "Relative descriptions relate one perceived geometric property to another (e.g., point a is twice as far away as point b). Ordinal descriptions are a special case of relative measure in which the sign, but not the

magnitude, of the relations is all that is represented" [24]. Our focus is on ordinal depth as absolute depth, for different datasets, can mean very different things yet the datasets can appear to be of similar size in a volume-rendered image.

3.1.1 Main Contributions

In this chapter, we study the use of depth of field (DoF) and its impact on the perception of ordinal depth in DVR to establish the conditions under which it is most beneficial. The DoF technique proposed by Schott et al. [1] was implemented in the SLIVR renderer of the VisIt visualization software [5]. The experiment is run on site to control experimental parameters such as screen quality, luminosity, and attention of subjects. Three hypotheses tested were:

1. HYP1: DoF will help improve the accuracy of ordinal depth perception in a volume-rendered image in which there are multiple features.
2. HYP2: DoF will help improve the speed of ordinal depth perception in a volume-rendered image in which there are multiple features.
3. HYP3: If users view a moving focal plane, correct perception of ordinal depth will improve.

To the best of our knowledge, this is the first comprehensive user study on the use of DoF as a depth cue in DVR. The main contributions of this project are:

- Establish whether DoF is a useful depth cue in volume-rendered images.
- Determine for which kind of datasets DoF is most helpful.

3.2 Depth of Field

The mechanics of DoF can be derived from the Thin-Lens equation:

$$\frac{1}{f} = \frac{1}{s} + \frac{1}{z_f} \tag{3.1}$$

where f is the focal length, s is the distance from the lens to the focal plane, and z_f is the distance from the lens to the object as shown in Fig. 3.3 (a). When light from a point in the scene passes through a camera lens, it should ideally be focused on a single point on the image plane. However, if the image plane is not in the correct position, the point is mapped to a circular region instead, c in Fig. 3.3 (a). The diameter of the region c can be determined according to equation 3.2:

$$c(z) = A\frac{|z - z_f|}{z} \tag{3.2}$$

where $c(z)$ is the diameter of the circle of confusion and A is the aperture of the lens as shown in Fig. 3.3 (b). Fig. 3.3 (b) also shows that on both sides of the focal distance z_f, we can have regions having similar diameters for the circle of confusion, which would translate to the same amount of blur in an image. The main difference between the regions, before and after the focus region, is the rate at which the amount of blur increases in each region. The rate at which blur increases in the region between the camera and focal length is much greater than the rate at which it increases after the focal length.

3.3 Depth of Field for DVR

DoF is implemented as shown in Fig. 3.4. A GPU slice texture-based volume renderer is used and the scene is broken down into two sections: before and after the focal plane. For the part between the focal plane and the front of the volume (where the camera is), the volume is processed from the front of the volume to the focal plane (in a front-to-back manner), and each slice is blurred according to its position. For the part behind the focal plane, the volume is processed from the end of the volume to the focal plane (in a back-to-front manner). Each rendered slice is blended with a blurred region of the previous slice. The blur kernel's size decreases as the slice approaches the focal plane. Two directions are needed to ensure that the in-focus region does not contribute any of the blurred regions. A more detailed description of the algorithm with a pseudocode can be found in [1].

3.4 User Study Setup

The aim of the experiment is to determine whether DoF provides a better understanding of the ordinal depth of different features in a volume-rendered image. More specifically, we want to be able to check these three hypotheses: 1) DoF helps improve the accuracy of the determination of ordinal depth; 2) DoF helps improve the speed of the determination of ordinal depth; 3) if users can change the position of the focal plane, correct perception of ordinal depth will improve.

To test the three hypotheses, we carry out a static experiment and a dynamic experiment. In the static part, we show the test subjects a number of images and we ask them to select which of two circled features (located at different depths in the image) is in front. Fig. 3.5 shows an example of this. In the dynamic part, we show a video of a DVR dataset where the focal plane sweeps from the front to the back and back to the front. Here again two features are circled at different depths, and the subject is asked to decide which one is in front. As discussed by Knill [62], the influence of depth cues varies depending on the task. A specific depth cue might be important for one task but not very relevant for another. Consequently, to make our experiment as general as possible, we decided to have minimal user interaction.

3.4.1 Stimuli Description

To generate the stimuli, DoF was implemented, as described in Section 3.2, in the SLIVR renderer of VisIt 2.4.2. PsychoPy 1.7.4 [63] was used to display the images and movies to the subjects and to collect their answers.

As can be seen from Fig. 3.6, the background color for each dataset is different. The background color, except for the flame dataset where the background color interfered too much due to the highly transparent nature of the image, was carefully chosen by computing the Michelson contrast as follows: the image is generated in VisIt with a background that can be easily removed from the image such as pure green (RGB: 0 1 0) or pure blue (RGB: 0 0 1) depending on the image. The Michelson contrast [64] is computed for all the colors that do not match the pure green or blue as follows:

$$M_c = \frac{L_{max} - L_{min}}{L_{max} + L_{min}} \tag{3.3}$$

where M_c is the Michelson Contrast, L_{max} is the maximum luminance, and L_{min} is the minimum luminance where luminance [65] is calculated as follows:

$$L = 0.2126 * Red + 0.7152 * Green + 0.0722 * Blue \tag{3.4}$$

where *Red*, *Green*, and *Blue* are the RGB components of the color. The background color is assigned to be a gray RGB value (same red, green, and blue) for which calculation of the Michelson contrast with the background is the same as calculating the Michelson contrast with only the dataset ignoring the background.

Five of the six datasets, shown in Fig. 3.6, were used for the first static part of the experiments for the following reasons: first, because of time constraints, we did not want the experiment to last too long but still manage to have enough data per dataset and, second, some features of the torso dataset would disappear when blurring was applied, and the subjects would see an empty ellipse or sometimes the feature behind the selected feature.

The datasets and their associated transfer function were selected so that a range of shapes were presented, some of which are familiar, such as the tree-shaped bonsai, and unfamiliar, such as the flame dataset, to the test subjects who were not regular users of volume rendering. Also, each resulting image generated has a different number and type of depth cue. The depth cues present are occlusion, perspective, relative size, familiar size, and texture gradient. For this test, we did not use shadows or changes in shading. The perspective projection settings is controlled so that all datasets have the same settings. Table 3.1 shows a taxonomy of the depth cues for the images in Fig. 3.6 that we used and Table 3.2 describes each of the images. The minimum separation between the features to be selected in an image was 5% of the whole depth of the volume and the maximum separation was 60%. The number of test images for each separation range is shown in Table 3.3.

Also, one of the characteristics of a good user study [66] is verifying that the participants are committed to answering truthfully, which can be accomplished in this case by verifying that perspective projection shows improvement compared to using orthographic projection. Perspective projection makes objects that are closer to the viewer appear larger than objects that are farther away, giving an additional depth cue.

3.4.2 Environment Setup

The experiment was conducted on site to ensure a similar environment setting for all participants. The study was carried out in a room where the curtains were closed and illuminated by white fluorescent tube light. Closing the curtains ensured that the lighting conditions did not vary during the day based on the position of the sun so that the test subjects do not experience different lighting conditions that might affect the colors on the computer screen. Moreover, the performance of the eye-tracker we used requires specific light condition for better tracking accuracy, which can be controlled with fluorescent light. Eye tracking is not the most important part of the study but was included to see if it would help us understand the results of the experiment. The T2T (Talk to Tobii) [67] package was used to interface with the Tobii T60 eye-tracker [68].

3.4.3 Apparatus

A Macbook Pro was used to run PsychoPy, which was connected to the eye-tracker. A gamepad was used as the input device as it is more ergonomic than a keyboard or mouse. On the gamepad, the test subjects pressed any of the left buttons to select the left feature and any of the right buttons to select the right feature. A simple and easy-to-use input interface is important since the subjects spent on average 30 minutes for the whole experiment and so we wanted to make it as easy and as less tiring for them as possible to try to minimize the impact of boredom and fatigue. Also, interaction with a gamepad is less likely to introduce perceptual bias as it offers a simple and straightforward method of entering selection.

3.4.4 Participants and Design

Twenty-five subjects (6 females and 19 males) participated in the user study. All test subjects had good eye sight or corrected vision; seven wore glasses and none were color blind. All but one test subject reported being right handed. The age range is shown in Table 3.4. All participants had some experience with computer graphics through games or movies, but most of them were not familiar with

volume rendering: none of them were students, researchers, or users developing or working with volume rendering.

A within-subject design was used in which all the participants completed all the tests. The experiment, carried out over 2 days with 25 participants, used a two-alternative forced-choice methodology [69]. Each participant spent approximately 30 minutes completing the experiment (along with calibrating the eye-tracker and training for the experiment) and each test was followed by a debriefing session.

3.4.5 Experimental Procedure

To test the three hypotheses, two experiments were carried out: a static experiment that featured only images and a dynamic experiment in which the focal plane was moving. Both experiments were conducted as follows:

1. Calibration of the eye-tracker: The subject was asked to stare at a red circle on the screen that appeared for 4 seconds at one position and then jumped to another position. The calibration stage lasted about 1 minute.
2. Training for the static part of the experiment: First, the overall task was explained to the user. The input device to be used, the gamepad, was introduced and the subject was briefed on how to use it. Next, seven training images were presented, four with DoF and three without. In each image, two features were circled with an ellipse, and the subject was asked to select which one is in front. We requested the correct answer for each before proceeding to the next image to ensure that the subject understood the task at hand. On completing this phase, we informed the subjects that they would now start with the experiments, but now they would not be required to give the correct answer to proceed to the next image.
3. The static experiment: Each of the 150 images was shown to the test subject who is asked to select which feature appears to be in front. The circle around the chosen feature changed color on being selected and the next image was presented. Response time and answers were recorded along with the eye tracking data.

4. Training for the dynamic experiment: A short animation describing the motion of the focal plane was presented to the user along with an explanation on what they have to do.
5. The dynamic experiment: Each of the 20 videos was shown and the test subject was asked to choose which of the two circled features appears to be in front. The answer was recorded along with the eye tracking data.
6. The experiment ended with a debriefing session during which the subject was asked for verbal feedback on the experiment, which was recorded by the experimenters.

3.5 Static Experiment

To test the first two hypotheses, whether DoF helps improve the speed and accuracy of perception of ordinal depth in a scene, we conducted the following experiment with 150 images of five datasets (aneurysm, backpack, bonsai, flame, and Richtmyer-Meshkov instability) under six different conditions:

1. orthographic projection
2. orthographic projection and front feature in focus (DoF Front)
3. orthographic projection and back feature in focus (DoF Back)
4. perspective projection
5. perspective projection and front feature in focus (DoF Front)
6. perspective projection and back feature in focus (DoF Back)

In each image, we presented the subject with two features. Each feature was surrounded by an ellipse and located at different horizontal positions so that it was clearly distinguishable which one was on the left and which one was on the right. The features were located at different depths, and participants were asked to choose the one they perceived as being in front. To select the left feature, the subject had to press on any of the left buttons on a gamepad, and vice versa for the right feature. Fig. 3.5 (a) shows an example image from the user study. Upon choosing a feature, the color of the circle changed. The order in which the different images were shown was randomized so as not to have all images under one specific condition or for a particular dataset following each other. However, all participants were shown the

test images in the same pre-randomized order. The submitted answer and the time taken was recorded for each test image.

3.5.1 Results

The average correctness and completion times recorded during the experiment were used to analyze the results.

The overall comparison of average correctness under the six conditions for all subjects reveals that there is a significant difference in the results for average correctness [one-way, repeated-measures ANOVA, $F(2.2,52.8) = 49.754$, $p < 0.001$]. Fig. 3.7 (a) shows the results for average correctness per task. For the individual datasets, when the average correctness under the six different conditions per dataset is compared, we see a statistically significant difference for average correctness of the results [two-way, repeated-measures ANOVA, $F(20,480) = 35.153$, $p < 0.001$]. Fig. 3.8 (a) shows the average correctness and Fig. 3.8 (b) the mean response time for each dataset. Running ANOVA for the response time shows that there is a low statistically significant difference between the means for perspective and perspective DoF front [one-way, repeated-measures ANOVA, $F(1,24) = 6.6$, $p < 0.017$]. Standard error is computed as follows: $\sigma/\sqrt{n}$ where n is the number of observations.

We had hoped the eye-tracker would show users trying to find the separation between the in-focus and out-of-focus region before making a decision. Unfortunately, for most test subjects, such a behaviur was not observed. We noticed that their gaze jumped from one of the circled features to the other. In some cases, their gaze would linger on some of the central figures, such as the canister in the backpack dataset, or they would try to follow the veins in the aneurysm dataset.

3.5.2 Discussion

The results show that, as expected, perspective projection is better than orthographic, but hypotheses 1 and 2 are not fully validated.

Perspective projection is expected to be better than orthographic projection because it causes objects in front to appear larger, and humans are used to having larger objects in front of smaller ones. This is confirmed in Fig. 3.8(a) where

the average correctness from perspective projection is higher than the average correctness from only ortographic projection.

The first hypothesis, DoF will help improve the accuracy of ordinal depth perception in a volume-rendered image where there are multiple features, is only partially validated. HYP1 is supported by the experiment if DoF is in front but is not supported by the experiment if DoF is on the back feature due, we believe, to depth cue conflict. Drascic et al. [70] reported that when depth cues provide conflicting information, there is an increase in uncertainty and a decrease in accuracy. Humans are used to seeing objects far away as blurred whereas those close to us are usually well defined. When DoF is on the back feature, the front feature appears blurry, contradicting what we are used to seeing. Boucheny et al. [9] reported a similar incident in their user study for volume rendering. In their first experiment, they had two cylinders arranged in different z depths, and the subjects were asked to state which one was in front of the other. Whenever the small cylinder was in front, the percentage of correct answers would drop drastically, from 70% to 30%, likely due to depth cue conflict since we are used to seeing objects close to us as big and far away objects as small.

The second hypothesis, DoF will help improve the speed of ordinal depth perception in a volume-rendered image where there are multiple features, is not fully validated. We saw that both accuracy and speed improved when DoF is applied on the front feature in all cases except the bonsai. The average correctness for the bonsai dataset increased from 57% for perspective projection with DoF to 82% for perspective projection with DoF front, but the participants took more time to make a decision, perhaps because they spent more time analyzing the image since they had more information at their disposal. The decrease in speed when DoF is on the back feature can be explained by depth cue conflict. The flame dataset is an exception to that. During debriefing, many participants found that the flame dataset, with or without DoF, was still extremely hard to understand, and so they often gave up trying to find the correct answer as they deemed the task too hard.

We also identified a pattern in our results that seems to be linked to the datasets. Namely, we saw that the Richtmyer-Meshkov instability and the bonsai seemed to

have similar results: both have a slight difference between perspective and DoF with perspective on the back feature, and a marked improvement for perspective with DoF front. We believe this pattern is the result of humans being familiar with the two shapes. The Richtmyer-Meshkov instability looks like a landscape (though it is a computational fluid dynamics dataset) and from the eye tracking data, we saw that the subjects' gaze swept through it as if it were a landscape. Moreover, during the debriefing, it was often referred to as a "landscape" by the participants. The bonsai looks like an ordinary tree. Therefore, familiarity with the shape helped the subjects in identifying the correct location of the features; familiarity is also one of the main depth cues people use to correctly determine depth. For response time, we saw that the response speed for perspective DoF front is higher for the Richtmyer-Meshkov than for any other dataset and condition.

The backpack dataset proved to be very challenging for the subjects to understand. We usually have a floor or ground in the real world that we use as frame of reference for the horizon, but here, everything appears to float in mid air. Perspective projection aided viewers as the closer small spherical-like objects appeared bigger than the remote ones. The result is strikingly similar to the aneurysm dataset. Both datasets have similar average correctness values for the perspective, perspective with DoF front and perspective with DoF back. The mean response times are quite similar except for DoF front. We believe that this is due to similarities between these two datasets: both lack a floor-like structure and have shapes that are not very common.

The flame dataset was the hardest for the subjects to understand. Adding DoF on the front feature helped increase correct perception from about 33% to 55%. However, when DoF was applied on the back feature, participants never answered correctly. The reason why the test subjects had so much trouble with this particular image is the amount of translucency present. If the flame dataset had been rendered using a transfer function that makes the dataset appear opaque, with distinct surfaces, we believe that the resulting volume rendered image would have been easier to understand. Humans perceive surfaces in everyday life, so understanding depth in these cases is quite easy. On the other hand, translucent

objects with fuzzy undefined surfaces are quite rare in everyday life, which could also be the reason why this image was hard to understand. Breaking down the results per dataset, we observe the same behavior: DoF on the front object helps reinforce the correct perception that an object is closer to the viewer.

3.6 Dynamic Experiment

To test the third hypothesis, whether being able to change the position of the focal plane will improve the correct perception of relative depth, a second experiment was carried out. To minimize user interaction, which could introduce bias in the experiment [71], we made a video of the focal plane sweeping from the front to the back and back to the front. This part of the experiment is referred to as the dynamic part of the experiment. We had 20 videos of the six datasets (aneurysm, backpack, bonsai, flame, Richtmyer-Meshkov instability, and thorax) each lasting approximately 17 seconds. We recorded only the answers, not the time, as the subjects are asked to watch each video completely before answering.

3.6.1 Results

Fig. 3.9 shows the average success rate for the different datasets. The accuracy rate is around 90% for the aneurysm, backpack, bonsai, and Richtmyer-Meshkov instability, but the accuracy for the translucent datasets of the flame and thorax is lower. From Fig. 3.8 (a), we see that having DoF in the dynamic part is not much better than DoF on the front feature except for the bonsai dataset. Doing an ANOVA to compare the means for DoF Front and dynamic [two-way, repeated-measures ANOVA, $F(2.28,68.2) = 1.58$, $p < 0.204$] reveals that there is no statistically conclusive difference between the means of these two. However, as shown in Fig. 3.10, the performance with videos seems to improve after repeated exposure. Note that time is not used here as a performance metric as the subjects were advised to watch the whole video, which evened out the response time.

3.6.2 Discussion

Hypothesis 3, if users view a moving focal plane, correct perception of ordinal depth will improve, is not validated by this experiment. From Fig. 3.8 (a), we see

that a static DoF focused on the front feature is often better than having a moving plane, except for the aneurysm and the bonsai. However, a video is always better than DoF on the back feature, which significantly degrades accuracy.

During the debriefing session, we found that the most accurate answers were from subjects who quickly understood that since the focal plane was sweeping from the front to the back and back to the front, the feature that appears not blurred first is the one in front. If they missed the ordering on the first pass, they would try to see it on the second pass. The less successful subjects usually had to see more videos to understand the mechanics of the moving plane; the front feature is the first that will appear in focus. Also, from the debriefing session, we learned that some subjects were focusing on one feature and did not see when the other feature was changing from out of focus to in focus, which might not have been the case if the subject could directly control moving the plane. The subjects would have been able to check exactly when each feature moved in and out of focus.

We also see that transparency is still a major issue even in the videos. For datasets that have hard surfaces, accuracy improves unlike for the flame and the thorax (translucent datasets), for which we still see a quite low accuracy rate. One of the issues with the translucent surfaces is they can disappear after DoF has been applied. For these datasets, we probably need a finer control over the focus to improve the results. Here again we have mixed results. With videos, DoF performance is more consistent; the subjects correctly identified which of the two circled features was the front one. However, this did not match our expectations of having better performance when compared to DoF front in static cases for all the datasets.

3.7 Guidelines

Based on what we have observed, we would recommend that to improve perception of ordinal depth, it is very important that DoF be used on the front object and not the back for static images. Using DoF on the front object successfully reinforces the correct perception of depth. If the feature that needs to be focused on is near the back of the volume, rotating the volume by 180 degrees about the

y-axis to bring that feature to the front might be worth considering. We would also encourage users to have as many depth cues as possible. In line with other research on depth perception, we see that the more depth cues we have, the easier it is for the users to correctly understand the arrangement of objects.

Using a video would be beneficial if this option is available, especially if the back feature needs to be in focus. The negative impact of having DoF on the back object that we notice in static images is reduced here as the users can see features moving in and out of focus. However, the users should be told exactly what to expect so that they are not surprised by an animation popping up. As shown in Fig. 3.10, results improve over time as the participants become familiar with the videos.

3.8 Summary

We have conducted a user study on the impact of DoF for ordinal depth perception in DVR. From our results, we see that using DoF on the front object reinforces correct perception of depth in DVR. However, putting DoF on the back object leads to depth cue conflicts, and the results are worse than not using DoF. Appropriate use of DoF provides a consequent improvement in terms of correct depth perception for general cases in DVR. For the dynamic part of the experiment, we saw a general overall improvement, though performance is still worse for highly translucent datasets.

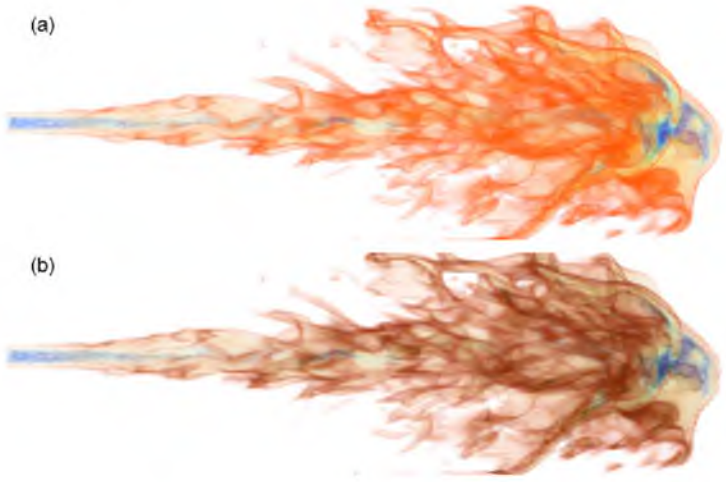

Fig. 3.1. Flame dataset: (a) original rendering and (b) with occlusion shading. We see that having ambient occlusion does not improve relative depth perception in this type of image.

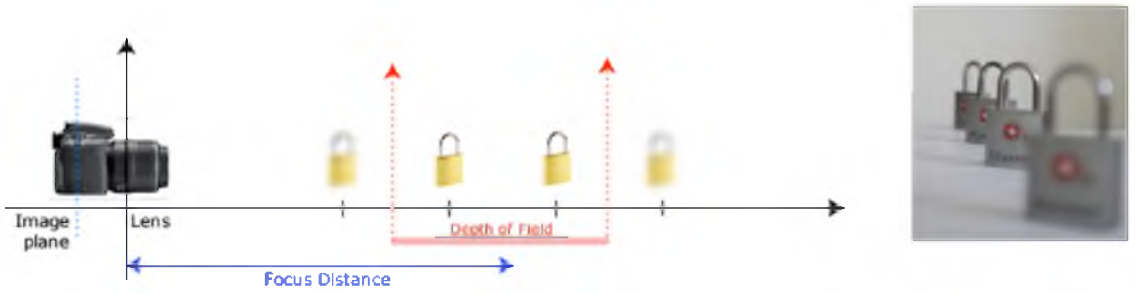

Fig. 3.2. Depth of field for camera imaging padlocks. Left: diagrammatic representation, right: actual photo.

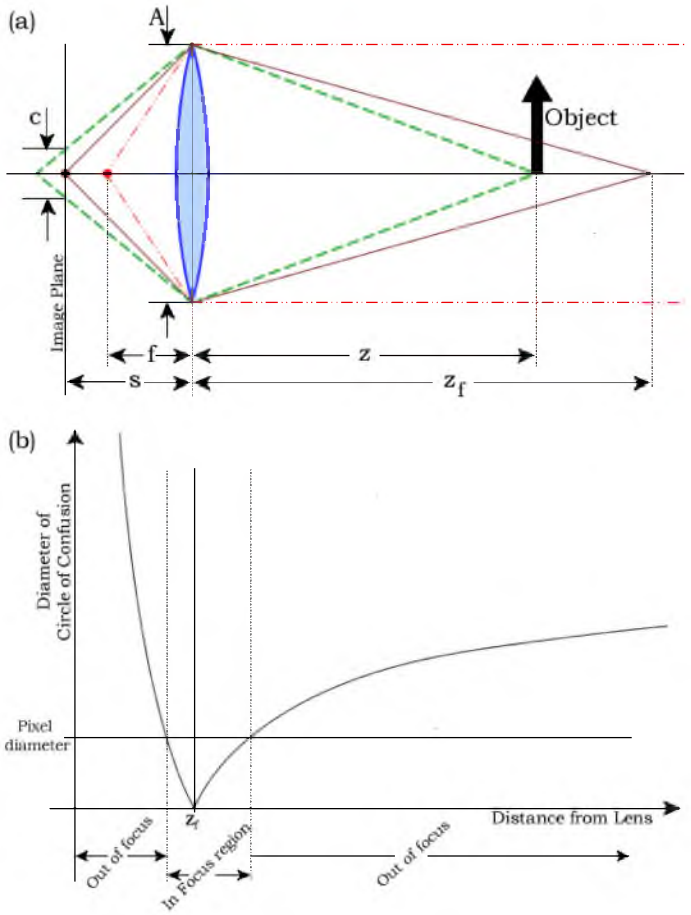

Fig. 3.3. Mechanics of depth of field: (a) lens setup and (b) circle of confusion.

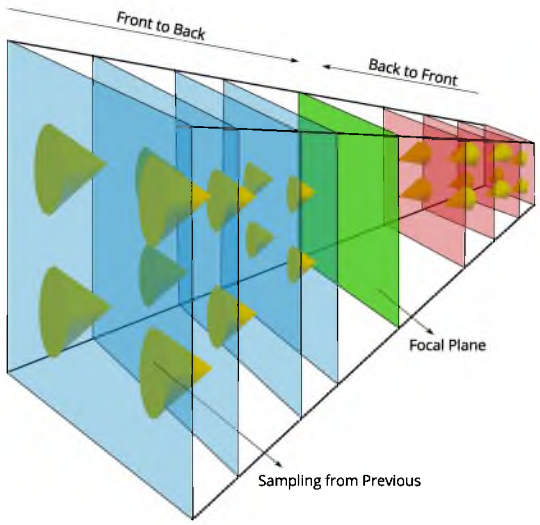

Fig. 3.4. Geometric setup of the DVR implementation of DoF

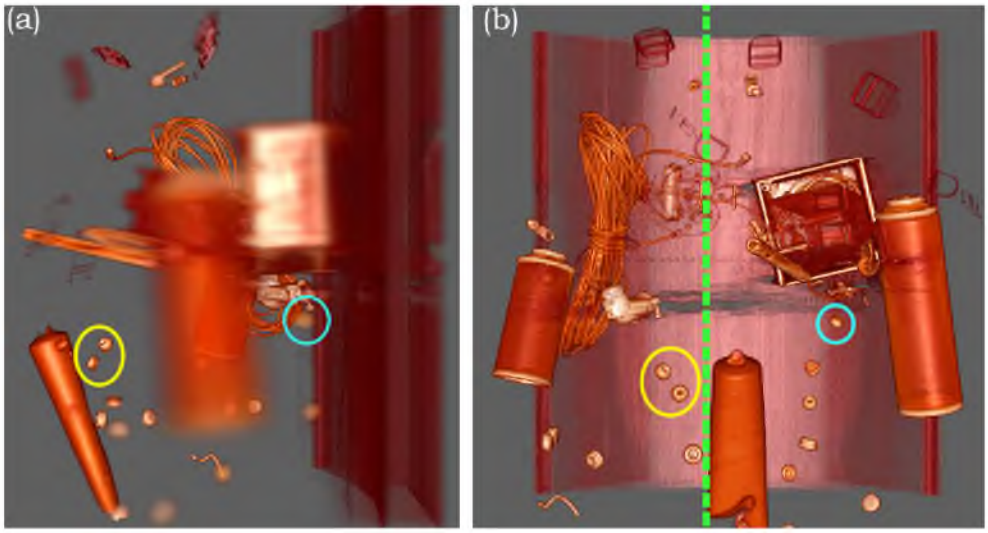

Fig. 3.5. The backpack dataset displayed from the side in (a) and from the front in (b) where the focus plane is shown as a dashed line. The features from which to choose from have been circled and we can see that the features are quite far apart.

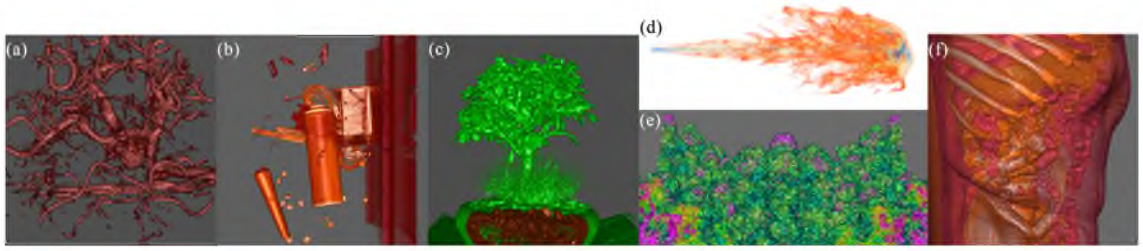

Fig. 3.6. The six datasets used: (a) aneurysm, (b) backpack, (c) bonsai, (d) flame, (e) Richtmyer-Meshkov instability, and (f) thorax.

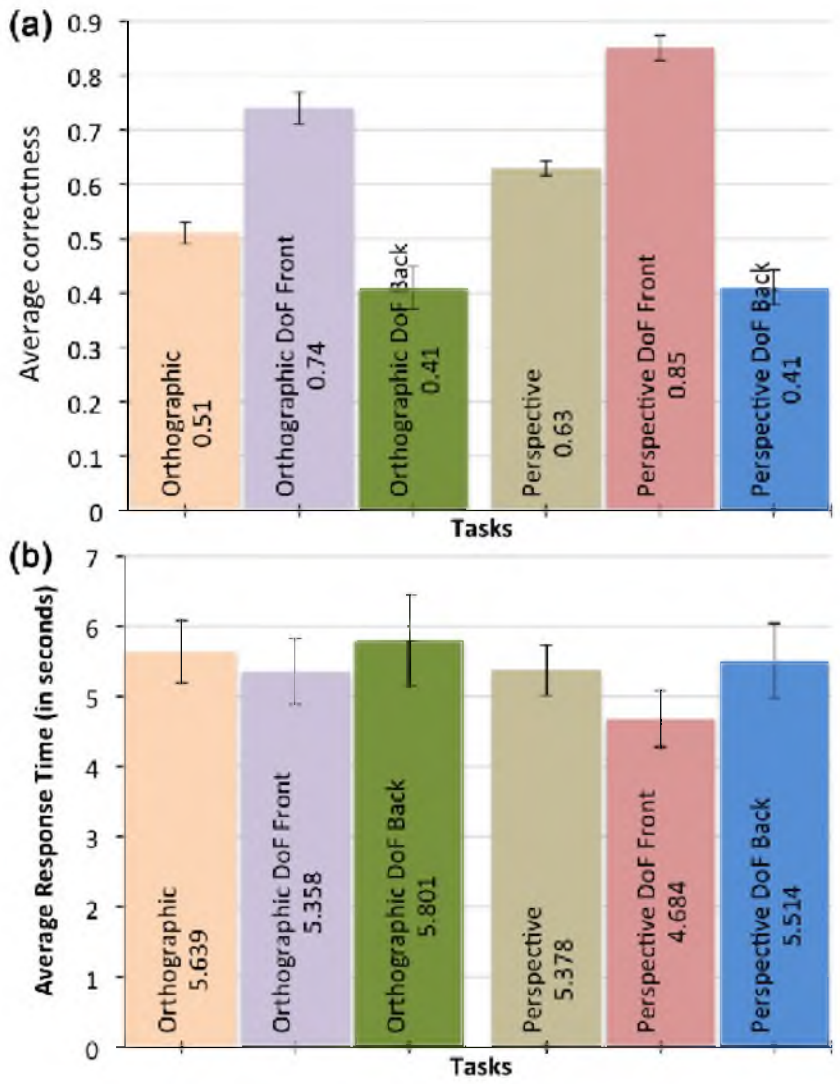

Fig. 3.7. Results for the static experiment: (a) average correctness for the different conditions (with static images) of the experiment with standard error, and (b) average response time for the different datasets and conditions (with static images) of the experiment with standard error.

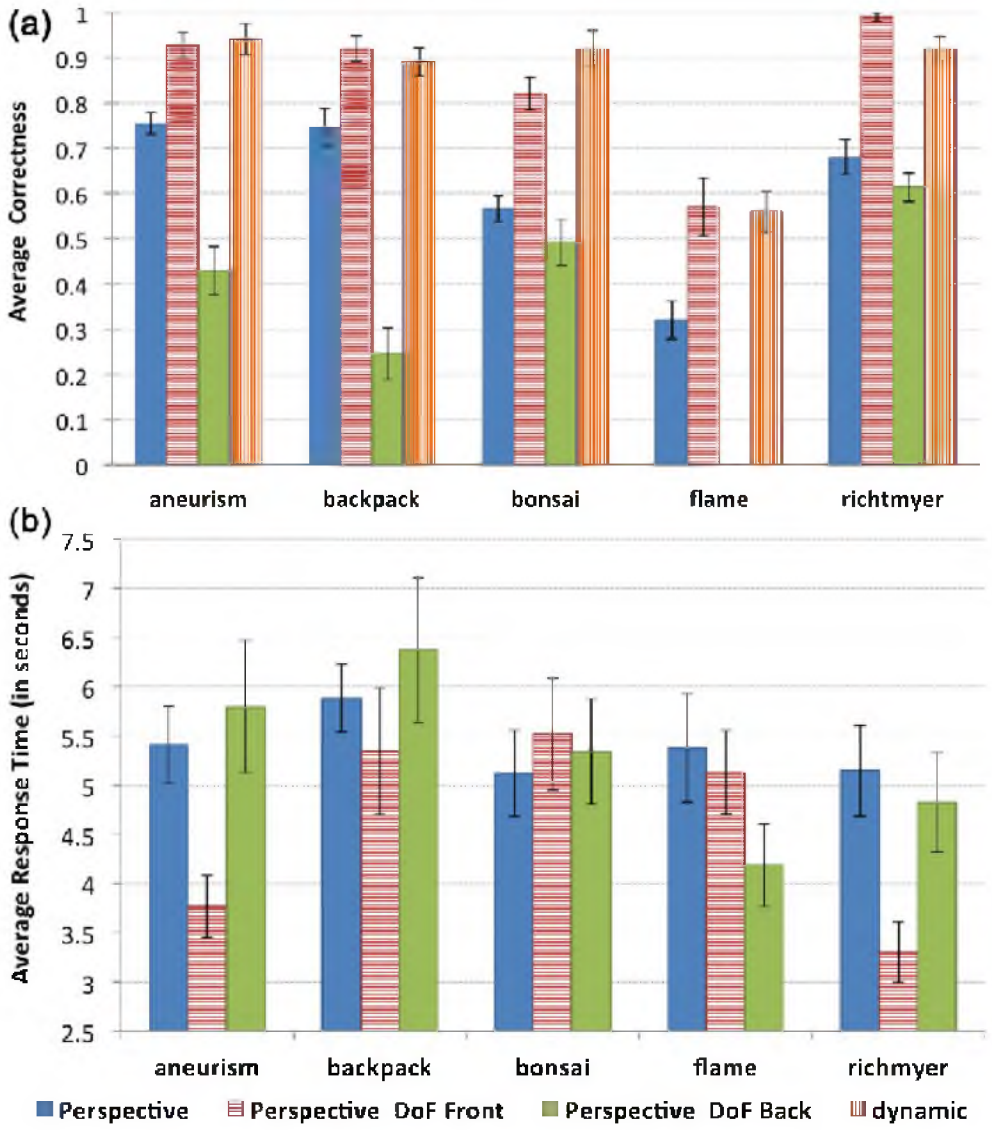

Fig. 3.8. Results for the static experiment per dataset: (a) average correctness for the different datasets, including static and dynamic, and (b) mean response time taken for the different datasets and conditions. Note: the 0 value for the flame dataset for perspective DoF Back indicates that all answers were wrong.

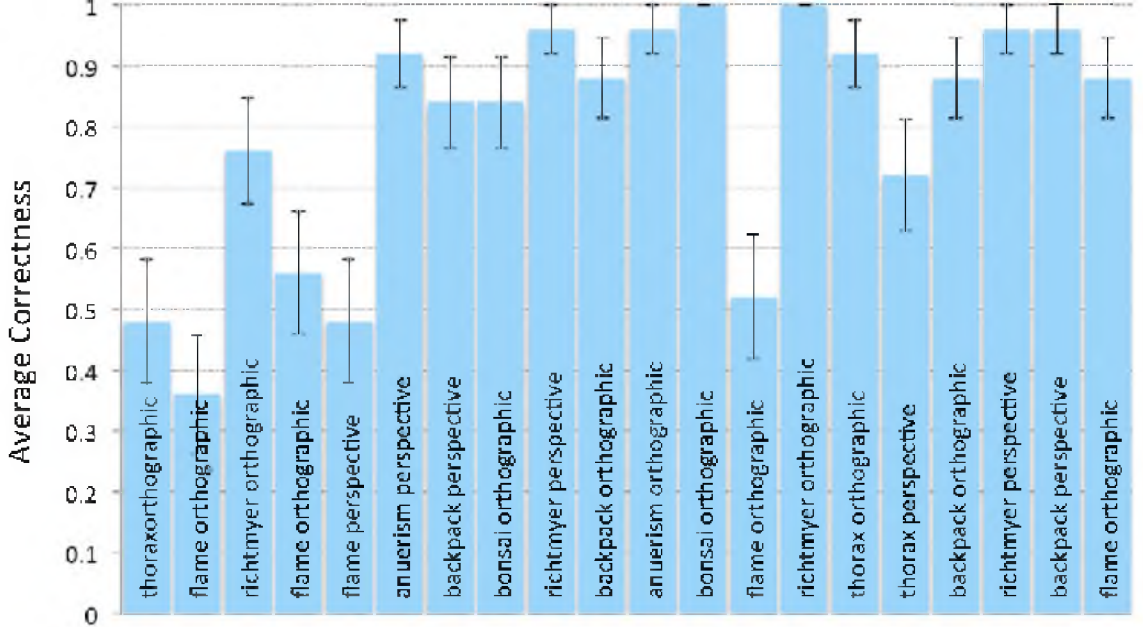

Fig. 3.10. The datasets from left to right shows the ordering in which the videos were shown to the participants.

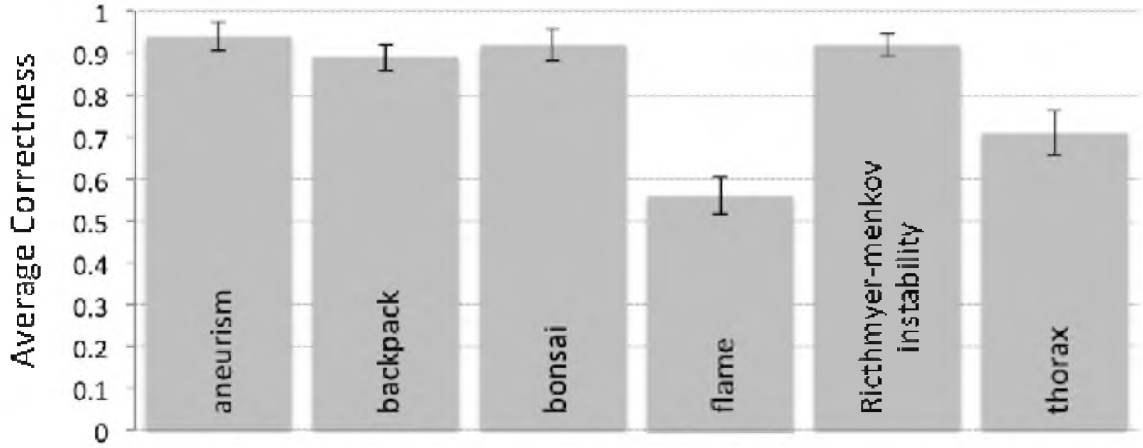

Fig. 3.9. Average correctness for the different datasets.

TABLE 3.1. Images and their associated depth cues. We have three levels for each: high, medium (Med), and low to indicate how useful each depth cue is expected to be in each volume-rendered image.

Dataset	Occlusion	Relative Size	Familiar Size	Texture Gradient
aneurism	High	Med	Med	Med
backpack	Low	Med	Low	Low
bonsai	High	High	High	High
flame	Low	Med	Low	Low
thorax	Med	Med	Low	Low
Richtmyer-Meshkov instability	Med	Med	Med	High

TABLE 3.2. Description of the images.

Image	Description
aneurysm	Has lots of occlusion that should be quite helpful to perceive depth, but the complex network of veins is quite confusing.
backpack	The ear buds are totally disconnected and appear to float in mid air, which is quite unfamiliar. Moreover, even for people who are familiar with depth of field, they normally see it as a progression over the image. This is not the case here. The background is not blurred, only the volume is. However, there is still some occlusion that could be helpful.
bonsai / Richtmyer-Meshkov instability	People are familiar with such shapes. The Richtmyer-Meshkov instability looks like a landscape (though it is not one), and everyone is familiar with seeing trees. Moreover, these shapes tend to have similar depth cues, as shown in Table 3.1.
flame	This is a combustion dataset that is extremely hard to understand. The chosen transfer function is what chemical engineers have used to visualize this data. One of their complaints was that it is very hard to understand this static image in a publication and so wanted to know if DoF could help in this case.
thorax	This image is complex due to the presence of many structures, some of which are quite transparent and faded completely when DoF was applied. We therefore used it only in the videos.

TABLE 3.3. The range of separation.

Amount of Separation	Number
0 - 10%	12
10 - 20%	23
20% - 30%	22
30% - 40%	37
40% - 50%	49
50% - 60%	7

TABLE 3.4. Age range of test subjects.

Age Range	Number
15 - 20	1
21 - 30	21
31 - 40	3

CHAPTER 4

TASK-OVERLAPPED DIRECT SEND TREE IMAGE COMPOSITING FOR HYBRID MPI PARALLELISM

4.1 Introduction

Supercomputers are now large distributed memory machines that have thousands of nodes, each with 10+ cores per node where each core has single instruction multiple data (SIMD) parallelism. Exchange of information inside a node is through shared memory, which is relatively fast, but exchange of information between nodes takes place though the interconnect, which is much slower than the compute capability of a node. One of the commonly cited challenges for exascale computing is to devise algorithms that avoid communication [72], as communication is quickly becoming the bottleneck. Consequently, the focus of algorithms for supercomputers is also to minimize communication.

Howison et al. [42] [14] found that using threads and shared memory inside a node and MPI for inter-node communication is much more efficient than using MPI for both inter-node and intra-node visualization. Previous research by Mallon et al. [73] and Rabenseifner et al., [74] summarized by Howison et al., indicates that the hybrid MPI model results in fewer messages between nodes and less memory overhead and outperforms MPI only at every concurrency level. Using threads and shared memory allows us to better exploit the power of these new very powerful multicore CPUs. Compositing algorithms, likewise, need to be changed. Instead of focusing on distributing the workload, the focus should now be now *minimizing communication*.

4.1.1 Main Contribution

The key contribution of this chapter is the introduction of Task Overlapped Direct send Tree, TOD-Tree, a new compositing algorithm for Hybrid/MPI parallelism that minimizes communication and focuses on overlapping communication with computation. There is less focus on balancing the workload and instead of many small messages, larger and fewer messages are used to keep the gathering time low as the number of nodes increases. We compare the performance of this algorithm with radix-k and binary-swap on an artificial and combustion dataset and show that we generally achieve better performance than these two algorithms in a hybrid setting.

4.2 Methodology

Since TOD-Tree has been tuned to work on hybrid MPI architectures, a process in our case is not a core but a node. At the start of the compositing phase, each node has an image that has been rendered from the part of the dataset it has loaded. Each image also has an associated depth from the viewpoint. Each node can know the depth of the images associated with other processes either through nodes sharing that information with each other or a k-d tree could determine the depth of every other node since it is often used to determine which part of a dataset a node should load, the latter could determine the depth of every other node. Each node sorts nodes by depth to know the correct order in which blending should be done. If the correct order is not used, the final image will not be correct. Also, from the extents of the dataset and the projection matrix used, it is easy to determine the height h, the width w, and the number of pixels p in the final image.

4.2.1 Algorithm

The TOD-Tree algorithm has three stages. The first stage is a grouped direct send. It is followed by a k-ary tree compositing stage. The display process then gathers data in the display stage. In all stages, asynchronous communication is used to overlap communication and computation. We first describe the algorithm conceptually.

Each node has a list of nodes sorted from smallest to largest depth. In the first stage, the nodes are arranged into groups of size r, which we will call a locality, based on their position in the depth-ordered list. Each node in a locality will be responsible for a region equivalent to $1/r$ of the final image. If r is equal to 4, there are four nodes in a locality, as shown in stage 1 of Fig. 4.1, and each is responsible for a quarter of the final image. The nodes in each locality exchange sections of the image in a direct send fashion so that at the end of stage 1, each node is authoritative on a different $1/r$ of the final image. The colors red, blue, yellow, and green in Fig. 4.1 represent the first, second, third, and fourth quarters of the final image on which each node is authoritative. Also in Fig. 4.1, there are 25 processes initially. In this case, the last locality will have five instead of four nodes, and the last node, colored orange in Fig. 4.1, will send its regions to the first r node in its locality but will not receive any data. In the second stage, the aim is to have only one node that is authoritative on a specific $1/r$ region of the final image. The nodes having the same region at the end of stage 1 are arranged in groups of size k based on their depth information. Each node in a group sends its data to the first node in its group, which blends the pixels, similar to k-ary tree compositing [39], [40], [12]. This stage can have multiple rounds. For example, in stage 2 of Fig. 4.1, six processes have the same quarter of the image, and therefore, two rounds are required until only one node is authoritative on a quarter of the image. Finally, these nodes blend their data with the background and send it to the display node, which assembles the final image, stage 3 in Fig. 4.1. We now describe in detail how we implement each stage of the algorithm, paying attention to the order of operation to maximize overlapping of communication with computation.

Algorithm 1 shows the setup for the direct send stage. There are a few design decisions to make for this part. Asynchronous MPI send and receive allows overlap of communication and computation. Posting the MPI receive before the send lets messages be received directly in the target buffer, instead of being copied to a temporary buffer and then copied to the target buffer. To minimize link contention, not all nodes try to send to one node. Depending on where they are in the locality,

Algorithm 1: Stage 1 - Direct Send

```
Determine the nodes in its locality
Determine the region of the image the node owns
Create a buffer for receiving images
Advertise the receive buffer using async MPI Recv
if node is in first half of locality then
    Send front to back using async MPI Send
else
    Send back to front using async MPI Send
Create a new image buffer
Initialize the buffer to 0
if node is in first half of region then
    Wait for images to come in front-to-back order
    Blend front to back
else
    Wait for images to come in back-to-front order
    Blend back to front
Deallocate receive buffer
```

the sending order is different. The buffer used as the sending buffer is the original image rendered in that node. To minimize memory use, there is only one blending buffer, and so the data must be available in the correct order for blending to start. The alternative would be to blend on the fly as images are received, but this requires creating and initializing many new buffers, which can have a very high memory cost when the image is large. The tests we carried out revealed that the gains in performance were not significant enough to outweigh the cost of allocating that much memory. The blending buffer also needs to be initialized to 0 for blending, which is a somewhat slow operation. To amortize this cost, blending is done after the MPI operations have been initialized so that receiving images and initialization can proceed in parallel.

The second stage is a k-ary tree compositing, shown in algorithm 2. Again, the receive buffer is advertised early to maximize efficiency. Another optimization that has been added is to blend with the background color in the last round while waiting for data to be received, thereby overlapping communication and computation. Also, alpha is needed when compositing but not in the final image. Therefore, while blending in the last round, the alpha channel is separated from the rest of the

image. It is still used for blending in that stage but is not sent in the gather stage, which allows the last send to be smaller and makes the gather faster.

Algorithm 2: Stage 2 - Tree Region

```
Determine if node will be sending or receiving
Create a buffer for receiving images
for each round do
    if sending then
        Send data to destination node
    else
        Advertise receive buffer using async MPI Recv
        if last round then
            Create opaque image for blending received images
            Create alpha buffer for blending transparency
            Blend current image with the background
            Receive images
            Blend in the opaque buffer
        else
            Receive images
            Blend in image buffer created in stage 1
Deallocate image buffer created in stage 1
Deallocate receive buffer
```

Algorithm 3: Stage 3 - Gather

```
Create empty final image
if Node has data then
    Send opaque image to display node
else
    if display node then
        Advertise final image as receive buffer
Deallocate send buffer from stage 1
```

Finally, the last stage of the algorithm is a simple gather from the nodes that still have data. Since the images have already been blended with the background in the previous stage, no computation is needed in this stage. The display node creates the final image, which is also the receive buffer, and indicates where data from each of the final senders should be placed. As soon as all images are in, compositing is

done. Also, at the end of this stage, the send buffer used in stage 1 is deallocated. Deallocation in earlier stages of the algorithm often involves waiting for images to be sent, but in stage 3, the images should have already been sent and so no waiting is required, which has been confirmed in some tests we carried out.

The two parameters to choose for the algorithm are the number of regions r and a value for k. r determines the number of regions into which an image is split for load balancing purposes. As the number of nodes increases, increasing the value of r results in better performance. k is used to control how many rounds the tree compositing stage has. It is usually best to keep the number of rounds low.

4.2.2 Theoretical Cost

We now analyze the theoretical cost of the algorithm using the cost model of Chan et al. [75], which has been used by Peterka et al. [13] and Cavin et al. [48]. Let the number of pixels in the final image be n, the number of processes be p, the time taken for blending one pixel be γ, the latency for one transfer be α, and the time for transferring one pixel be β. Stage 1 is essentially several direct sends. The number of sends in a group of size r per process is $(r-1)$ and the number of compositings is $r-1$. Since each of the r groups will do the same operation in parallel, the cost for stage 1 is determined according to equation 4.1:

$$(r-1)[(\alpha+\frac{n}{r}\beta)+\frac{n}{r}\gamma] \tag{4.1}$$

The second stage is a k-ary tree compositing. r tree compositings are taking place in parallel. Each tree has p/r processes to composite. The number of rounds is $log_k(p/r)$. For each round, there are at most $k-1$ sends. The cost for the k-ary compositing is determined according to equation 4.2:

$$log_k\frac{p}{r}[(k-1)[(\alpha+\frac{n}{r}\beta)+\frac{n}{r}\gamma]] \tag{4.2}$$

The cost for the final gather stage is determined according to equation 4.3:

$$(\alpha+\frac{n}{r}\beta) \tag{4.3}$$

The final total cost is as shown in equation 4.4:

$$(2r + (k-1)log_k\frac{p}{r} - 1)(\alpha + \frac{n}{r}\beta) + (r + (k-1)log_k\frac{p}{r} - 1)\frac{n}{r}\gamma \tag{4.4}$$

The cost for the other compositing algorithms: direct send, equation 4.5; binary swap, equation 4.6 and, radix-k, equation 4.7, quoted from the work of Cavin et al. [48], are shown below.

$$\alpha(p-1) + [(\beta+\gamma)n(\frac{p-1}{p})] \tag{4.5}$$

$$\alpha log_2 p + [(\beta+\gamma)n(\frac{p-1}{p})] \tag{4.6}$$

$$\alpha\sum_{i=1}^{i=r}(k_i - 1) + [(\beta+\gamma)n(\frac{p-1}{p})] \tag{4.7}$$

where k_i is each vector for each of the r rounds of radix-k.

These equations are useful but fail to capture the overlap of communication and computation. It is hard to predict how much overlap there will be as communication depends on the congestion in the network, but from empirical observations, we have seen that the equation acts as an upper bound for the time that the algorithm will take. For example, the total time taken for 64 nodes on Edison was 0.012 seconds for a 2048x2048 image (64MB). We now calculate the time using the equation and performance values for Edison on the NERSC website [76]: α is at least $0.25x10^{-6} seconds$ and the network bandwidth is about 8GB/s, so for one pixel (four channels each with a floating point of size 4 bytes) $\beta = 1.86x10^{-9} seconds$. The peak performance is 460.8 Gflops/node, so $\gamma = 8.1x10^{-12} seconds$. The theoretical time should be around 0.015 seconds. The model effectively gives a maximum upper bound for the operation, but more importantly, this calculation shows how much time we are saving by overlapping communication with computation. In the tests we carried out, we never managed to get 8GB/s bandwidth; we always got less than 8GB/s, and yet the theoretical value is still greater than the actual value we are measuring.

Fig. 4.2 shows the profile for the algorithm using an internally developed profiling tool. All the processes start with setting up buffers and advertising their

receive buffer, which is shown colored yellow in the diagram. This step is followed by a receive/waiting to receive section, colored blue, and blending section, colored red. All receive communication is through asynchronous MPI receive whereas the sends for stage 1 are asynchronous and the rest are blocking sends. The dark green represents the final send to the display node, and the dark blue indicates the final receive on the display node. As can be clearly seen, most of the time is being spent communicating or waiting for data to be received from other nodes. A breakdown of the total time spent by 64 nodes on Edison is shown in Fig. 4.3.

As previously mentioned, the most time-consuming operations are send and receive, which is one of the reasons why load balancing is not as important anymore, and using tree style compositing is not detrimental to our algorithm.

4.3 Testing and Results

We have compared our algorithm against radix-k and binary-swap from the IceT library [41]. We are using the latest version of the IceT library, from the IceT git repository, as it has a new function icetCompositeImage, which, compared to icetDrawFrame, takes in images directly and is thus faster when provided with prerendered images. This function should be available in future releases of IceT.

4.3.1 Test Setup

The two systems that have been used for testing are the Stampede supercomputer at TACC and the Edison supercomputer at NERSC. Stampede uses the Infiniband FDR network and has 6,400 compute nodes that are stored in 160 racks. Each compute node is an Intel SandyBridge processor that has 16 cores per node for peak performance of 346 GFLOPS/node [77]. Since IceT has not been built to take advantage of threads, we did not build with OpenMP on Stampede. Both IceT and our algorithm will be compiled with g++ and O3 optimization. Edison is a Cray X30 supercomputer that uses the dragonfly topology for its interconnect network. The 5,576 nodes are arranged into 30 cabinets. Each node is an Intel IvyBridge processor with 24 cores and has a peak performance of 460.8 GFLOPS/node [76]. To fully utilize a CPU and be as close as possible to its peak performance, both threads and vectorization should be used. Both SandyBridge and IvyBridge processors have

256 bit wide registers that can hold up to eight 32 bit floating points; only when doing eight floating point operations on all cores can we attain peak performance on one node. Crucially, IvyBridge processors offer the vector gather operation, which fetches data from memory and packs them directly into SIMD lanes. With newer compilers, this can improve performance dramatically. On Edison, we fully exploit IvyBridge processors using OpenMP [78] and auto-vectorization with the Intel15 compiler.

The two datasets used for the tests are shown in Fig. 4.4. The artificial dataset is a square block where each node is assigned one subblock. The simulation dataset is a rectangular combustion dataset where the bottom right and left are empty. The artificial dataset is a volume of size 512x512x512 voxels, and the images sizes for the test are 2048x2048 pixels (64MB), 4096x4096 pixels (256MB), and 8192x8192 pixels (1GB). The combustion dataset is a volume of size 416x663x416 voxels. For the image size, the width has been set to 2048, 4096, and 8192. The heights are 2605, 5204, and 10418 pixels, respectively.

On Edison at NERSC, we were able to get access to up to 4,096 nodes (98,304 cores), whereas on Stampede at TACC, we have been granted access to a maximum of 1,024 nodes (16,384 cores). In the next section, we will show the performance for these two cases. Each experiment is run 10 times, and the results are the average of these runs after some outliers have been eliminated.

4.3.2 Scalability on Stampede

When running on Stampede, threads are not being used for the TOD-Tree algorithm. Both IceT and our implementation are compiled with g++ and O3 optimization. This is done to keep the comparison fair and also to point to the fact that it is the overlapping of tasks rather than raw computing power that is most important here. Also, we are not using any compression as most image sizes used are small enough that compression does not make a big difference. At 8192x8192 pixels, an image is now 1GB in size and having compression would likely further reduce communication.

Fig. 4.5 shows the strong scaling results for artificial data on Stampede. The TOD-Tree algorithm performs better than binary-swap and radix-k. The sawtooth-like appearance can be explained by the fact that we use the same value of r for pairs of time steps: r=16 for 32 and 64 nodes, r=32 for 128 and 256, and r=64 for 512 and 1024, and only 1 round for the k-ary tree part of the algorithm. Thus with r=32, for 256 nodes, there are eight groups of direct send, but there are only four groups of direct send at 128 nodes. Therefore, the tree stage must now gather from seven instead of from three processes and so the time taken increases. Also, this means that instead of waiting for three nodes to complete their grouped direct send, now the wait is for seven nodes. Increasing the value of r helps balance the workload in stage 1 of the algorithm and reduces the number of nodes that have to be involved in the tree compositing, and hence decreases the sending.

For images of size 2048x2048 pixels, compositing is heavily communication bound. As we increase the number of nodes, each node has very little data and so all three algorithms surveyed perform with less consistency as they become more communication bound and so more affected by load imbalance and networking issues. Communication is the main discriminating factor for small image sizes. For 8192x8192 images, there is less variation as compositing is more computation bound. Also, at that image size, IceT's radix-k comes close to matching the performance of our algorithm. On analyzing the results for TOD-Tree, we saw that the communication, especially in the gather stage, was quite expensive. Whereas a 2048x2048 image is only 64 MB, a 8192x8192 image is 1GB and transferring such big sizes is expensive without compression, which is where IceT's use of compression for all communication becomes useful.

In the test case above, we used only one round for the tree compositing. For large node counts, more rounds could be used. Fig. 4.6 shows the impact of having a different number of rounds for large node counts. For 256 nodes, there is an improvement of 0.018 seconds, but it is slower by 0.003 seconds for 512 nodes and 0.007 seconds for 1024 nodes. Having several rounds barely slows down the algorithm and can even speed up the results.

Fig. 4.7 shows the results for the combustion dataset on Stampede. One of the key characteristics of this dataset is that at the bottom, there are empty regions. This creates load imbalances. Also, the dataset is rectangular and not as uniform as the artificial dataset, but it resembles more closely what users are likely to be rendering. The load imbalance creates some different situations from the regular dataset that affects the IceT library a bit more than it affects the TOD-Tree compositing. This is because both binary-swap and radix-k give a greater importance to load balancing and if the data is not uniform, they are likely to suffer from more load imbalances. The TOD-Tree algorithm does not place that much importance on load balancing.

4.3.3 Scalability on Edison

On Edison, we managed to scale up to 4,096 nodes. The same values of r and k were used for up to 1024 nodes, as were used on stampede. These velues are: r=16 for 32 and 64 nodes, r=32 for 128 and 256 and, r=64 for 512 and 1024, r=128 for 2048 and 4096, and only 1 round was used for the k-ary tree part of the algorithm. The results for strong scaling are shown in Fig. 4.8. The performance of IceT's binary-swap was quite irregular on Edison. For example, for the 4096x4096 image, the time taken would suddenly jump to 0.49 seconds after being similar to radix-k for lower node counts (around 0.11 s). We therefore decided to exclude binary-swap from these scalings graphs. The sawtooth pattern is similar to what we see on Stampede for TOD-Tree. Both TOD-Tree and radix-k show less consistency on Edison compared to Stampede. On Edison, 8192x8192 images at 2048 and 4096 nodes are the only instances where radix-k performed better than the TOD-Tree algorithm. Again, the main culprit was communication time and TOD-Tree not using compression. In the future, we plan to extend TOD-Tree to have compression for large image sizes.

Furthermore, to try to better understand the cause of the sawtooth-shaped scaling, the TOD-Tree algorithm was modeled by plugging in the latency and bandwidth of the Cray X30 internonnect, and the FLOPS of the Ivy Bridge CPU of the Edison supercomputer into equation 4.4. The result, for a 4Kx4K image, is shown in Fig. 4.9. As we can see, the theoretical model predicts a sawtooth

appearance which is reflected in the actual runs that were performed on Edison and Stampede. This also means that better values of r and k can be chosen to try to minimize this sawtooth appearance.

4.3.4 Stampede versus Edison

Fig. 4.10 shows the result of the TOD-Tree algorithm on Stampede and Edison. The values of r used are the same as on Stampede for up to 1024 nodes. For 2048 and 4096 nodes, we set r to be 128. As expected, the algorithm is faster on Edison than on Stampede: the interconnect is faster on Edison and the nodes have better peak flop performance. On Stampede, we are not using threads, but on Edison, we are using threads and vectorization. The gap between the performance is bigger for low node counts, as each node has a larger chunk of the image to process when few nodes are involved and so a faster CPU makes quite a big difference. As the number of nodes increases, the data to process decreases and so the difference in computing power is less important as the compositing becomes communication bound. The sawtooth appearance is present in both but is amplified for Edison. On average, we are still getting about 16 frames per second for a 256MB images (4096x4096). At 2048 nodes on Edison, the time taken for TOD-Tree decreases, as can be seen in the middle chart of Fig. 4.8.

Fig. 4.11 shows the equivalent comparison but for 8192x10418 images using the combustion dataset. It is interesting to note that image compositing - using TOD-Tree - on Edison is initially much faster than on Stampede, but at 1024 nodes, the difference in time is negligible. When few nodes are used, a great deal of computation is required, as each node has a larger image to process and so having a powerful CPU is beneficial, but when there is less computation to do, the difference in computation power is no longer that important. IceT performs less consistently for this dataset probably because of the load imbalance inherent in the dataset.

4.4 Summary

In this chapter, we have introduced a new compositing algorithm for hybrid OpenMP/MPI Parallelism and shown that it generally performs better than the two leading compositing algorithms, binary-swap and radix-k, on the hybrid

programming environment. When using the hybrid parallelism, there is a quite a large difference between the computation power available to one node compared to the speed of inter-node communication. Hence, the algorithm must pay much more attention to communication than to computation if we are to achieve better performance at scale.

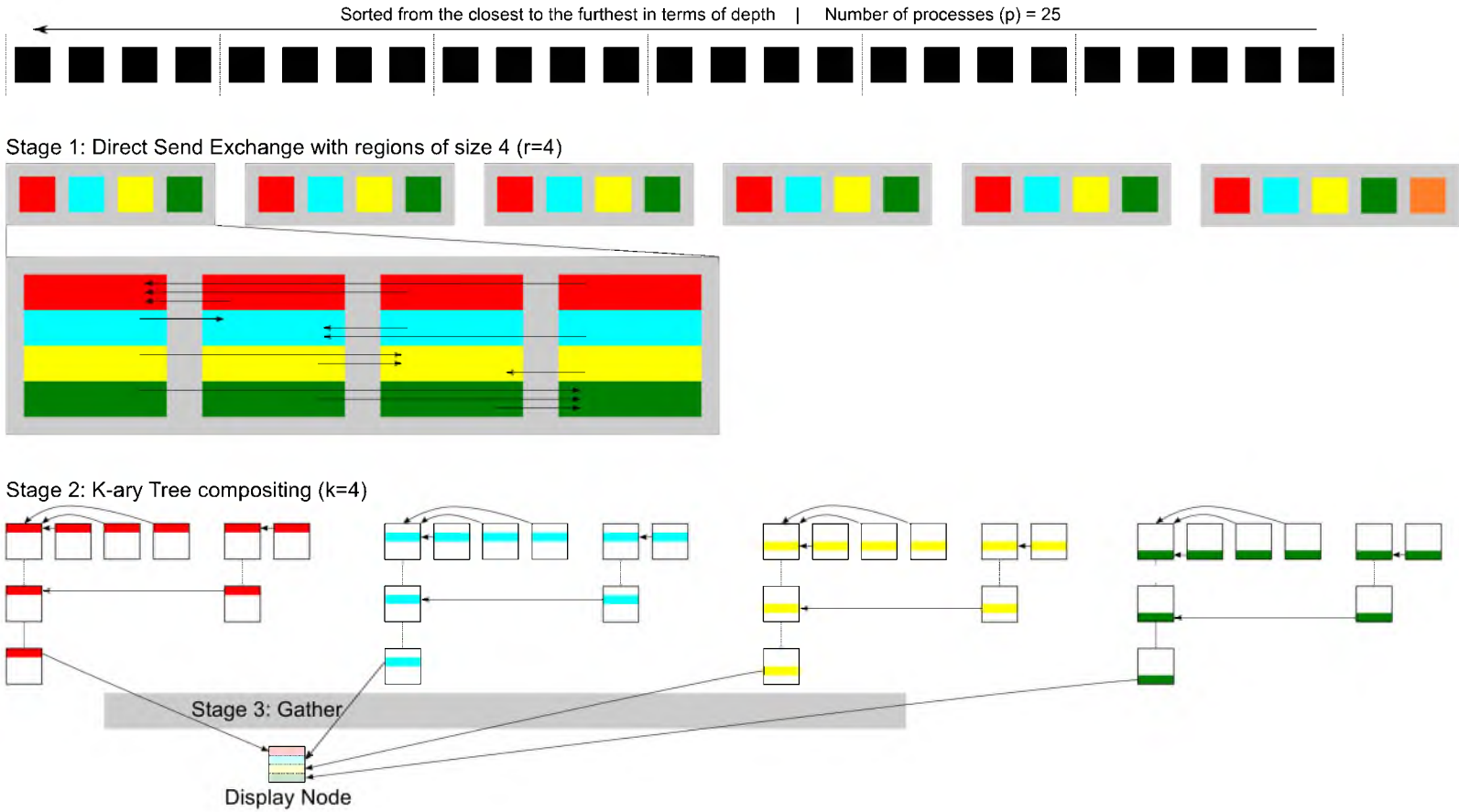

Fig. 4.1. The three stages of the compositing algorithm with r=4, k=4, and the number of nodes p=25. Red, blue, yellow, and green represent the first, second, third, and fourth quarter of the image.

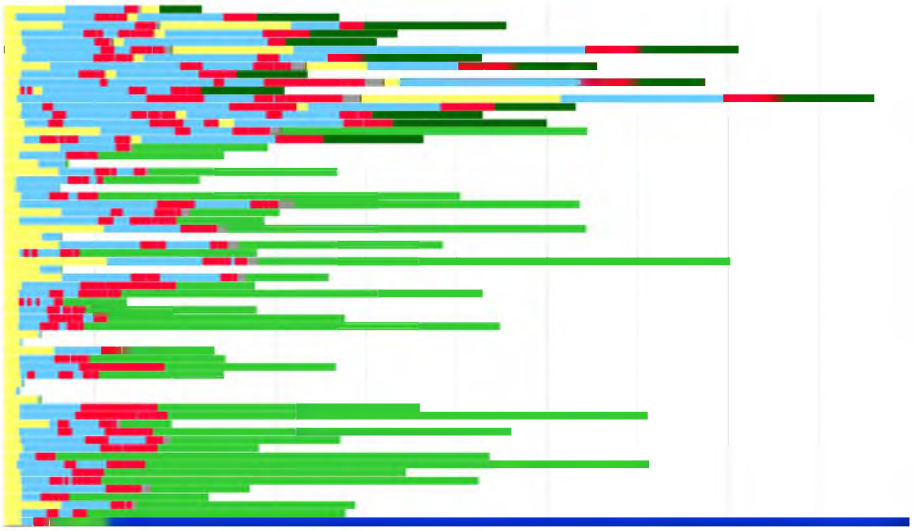

Fig. 4.2. Profile for 64 nodes for 2048x2048 (64MB) image on Edison at NERSC with r=16, k=8. Red: compositing, green: sending, light blue: receiving, dark blue: receiving on the display process. Total time: 0.012s.

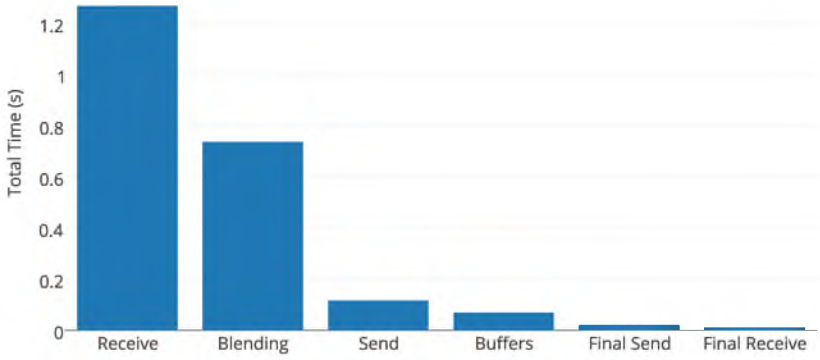

Fig. 4.3. Breakdown of different tasks in the algorithm.

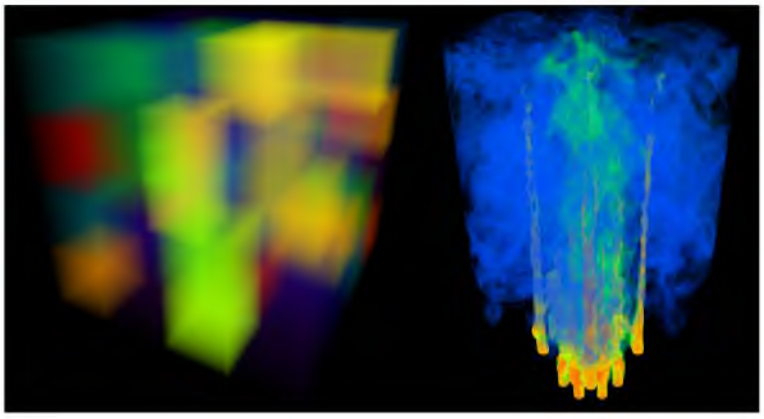

Fig. 4.4. The two test datasets used for testing: a synthetic dataset on the left and a combustion dataset on the right.

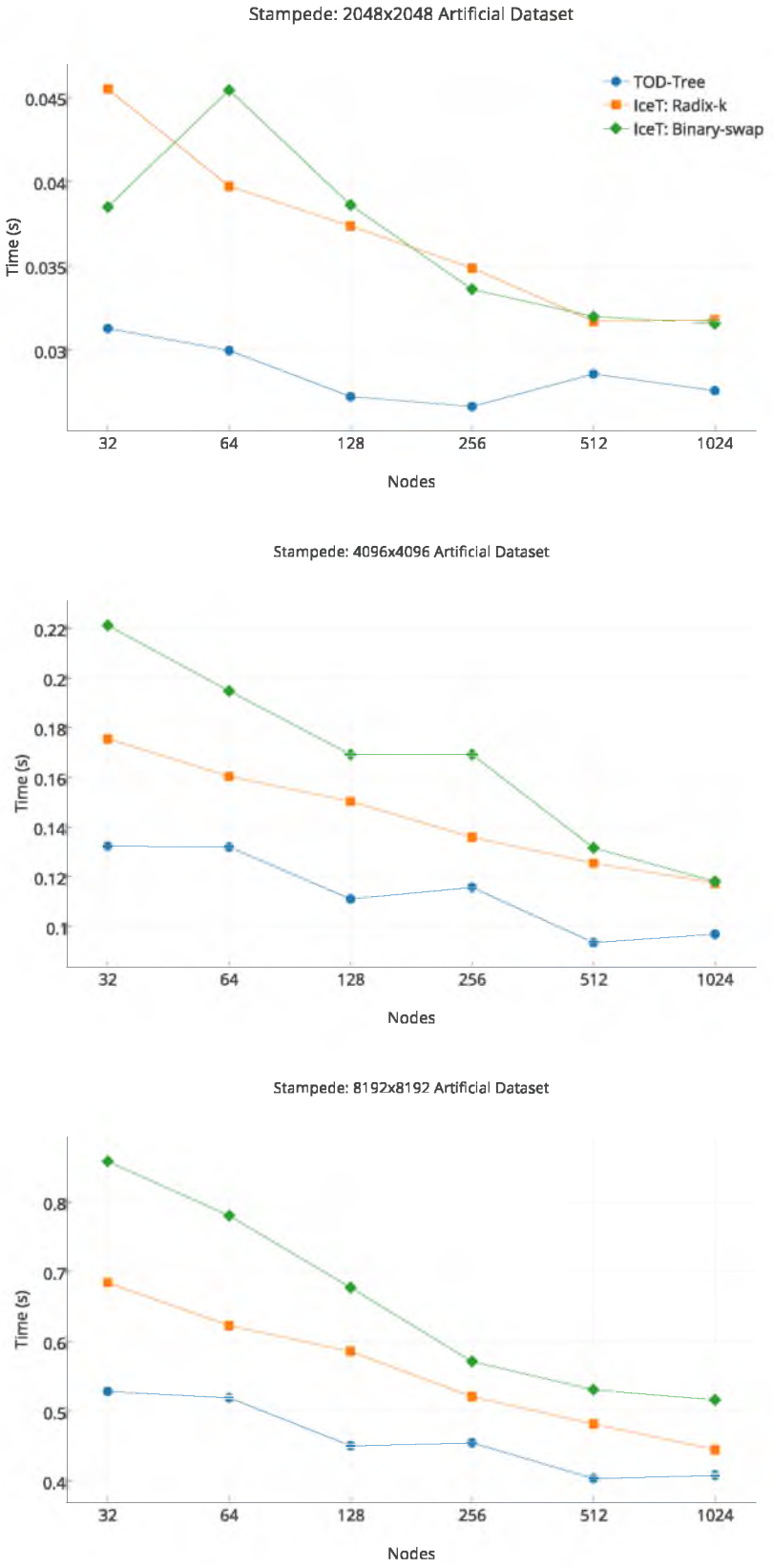

Fig. 4.5. Scaling for the artificial data on Stampede.

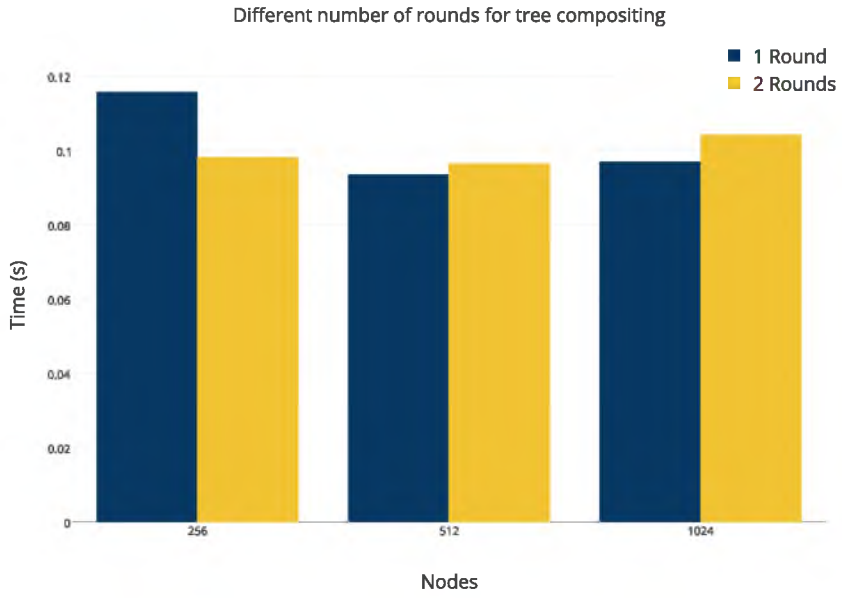

Fig. 4.6. Varying number of rounds for the artificial dataset for 4096x4096.

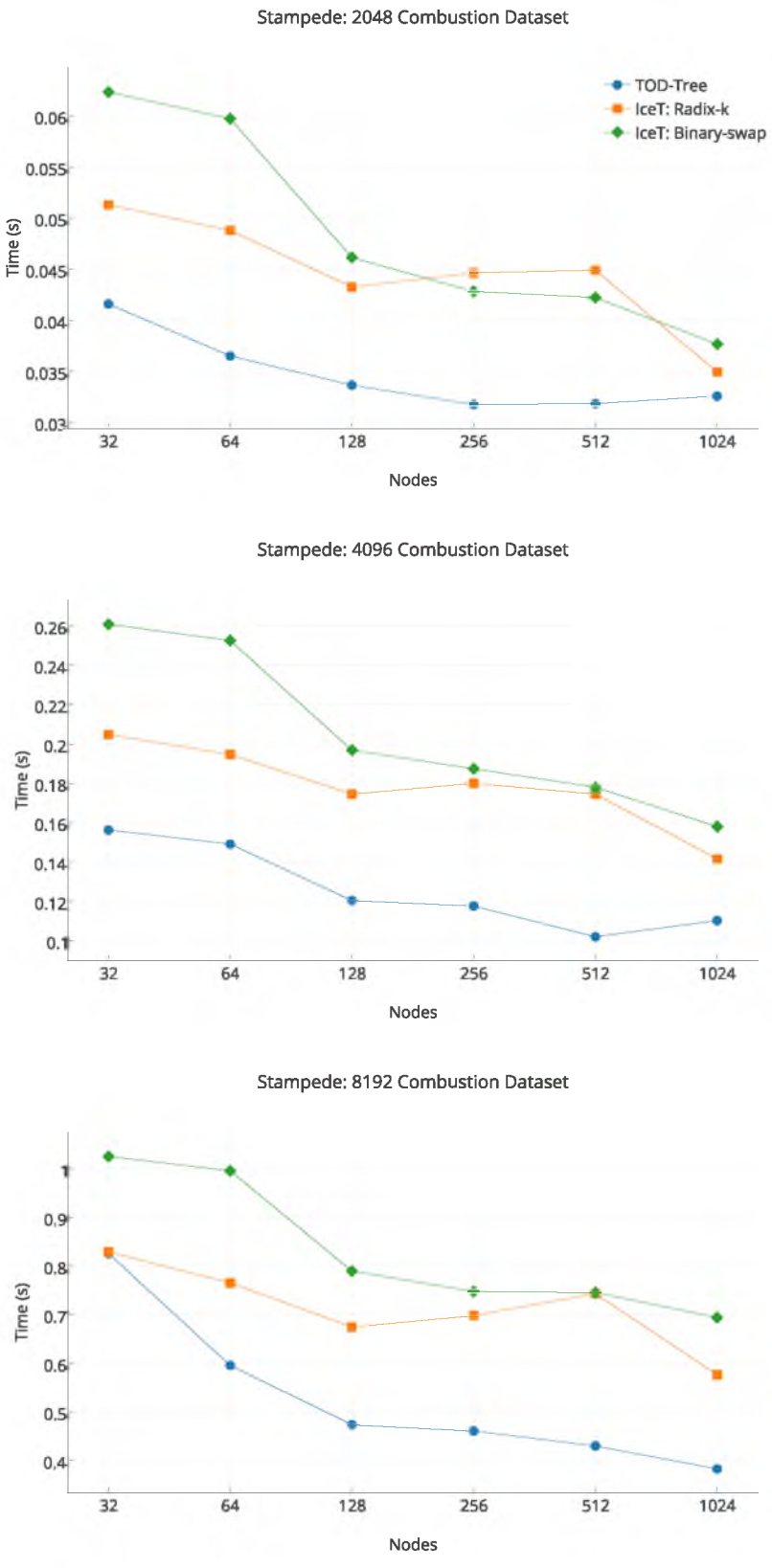

Fig. 4.7. Scaling for combustion data on Stampede.

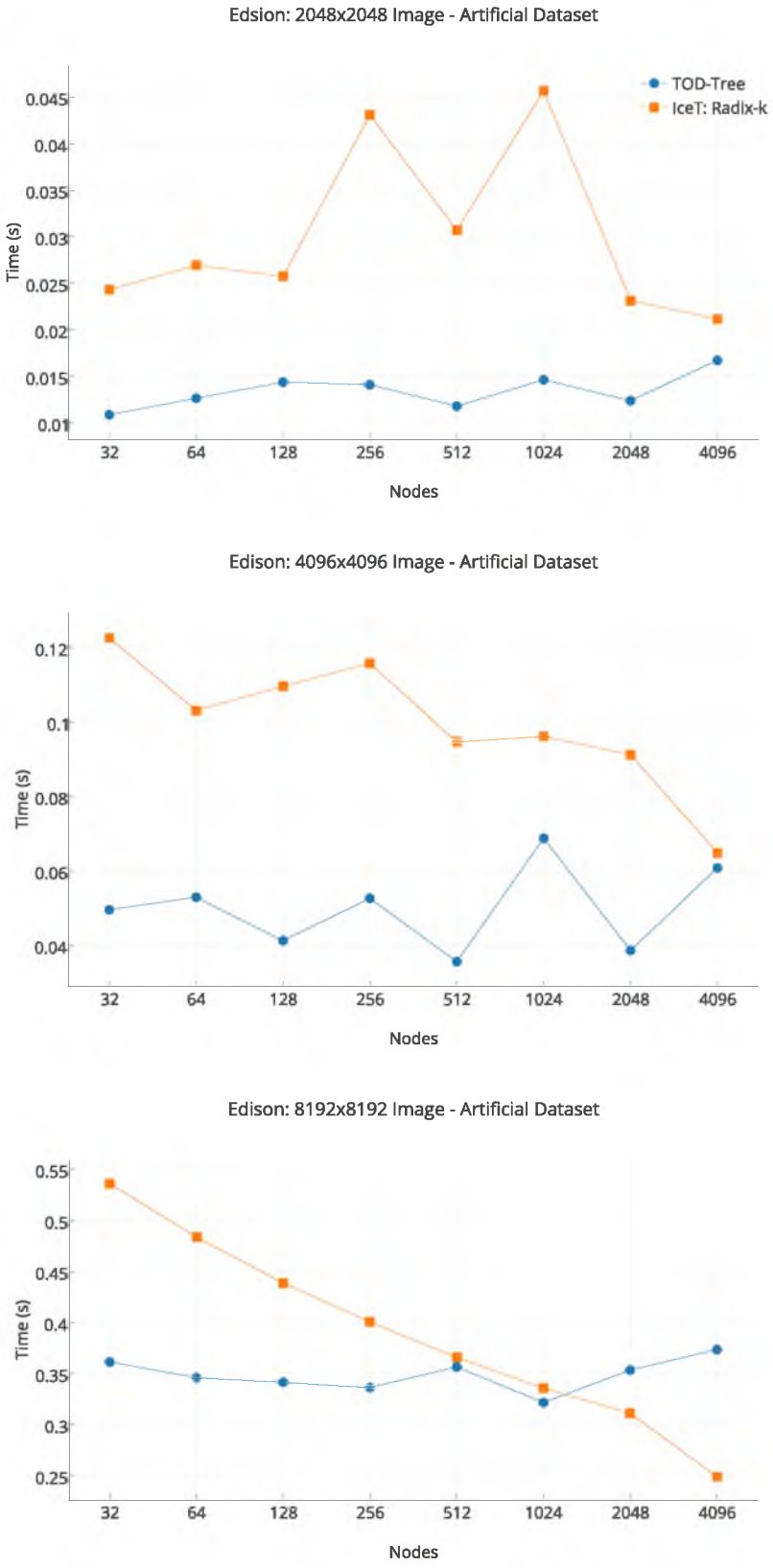

Fig. 4.8. Scaling for artificial dataset on Edison.

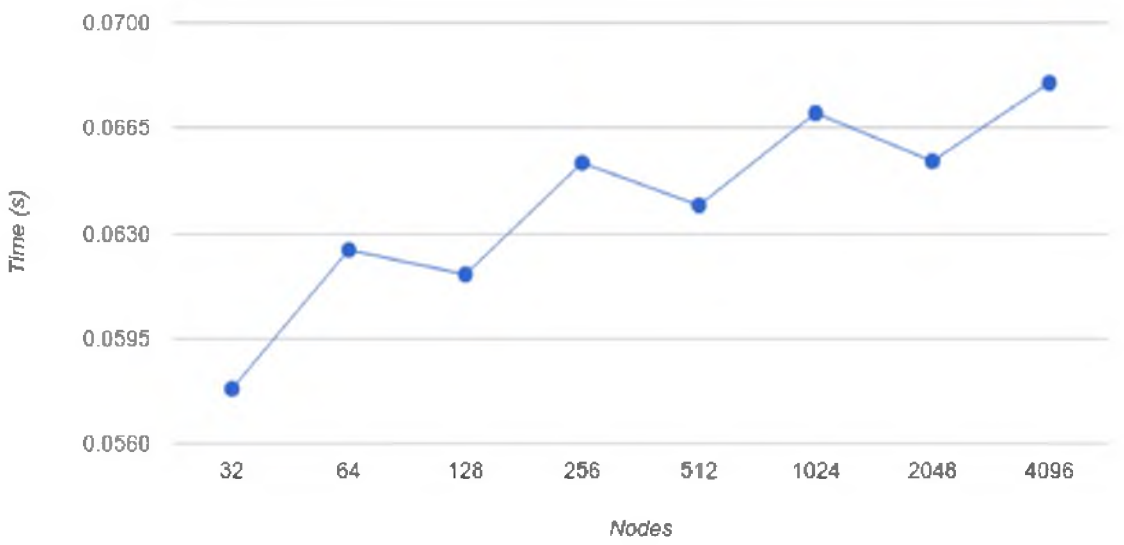

Fig. 4.9. Modelling TOD-Tree using the network latency, bandwidth, and compute capability of Edison for 4Kx4K images.

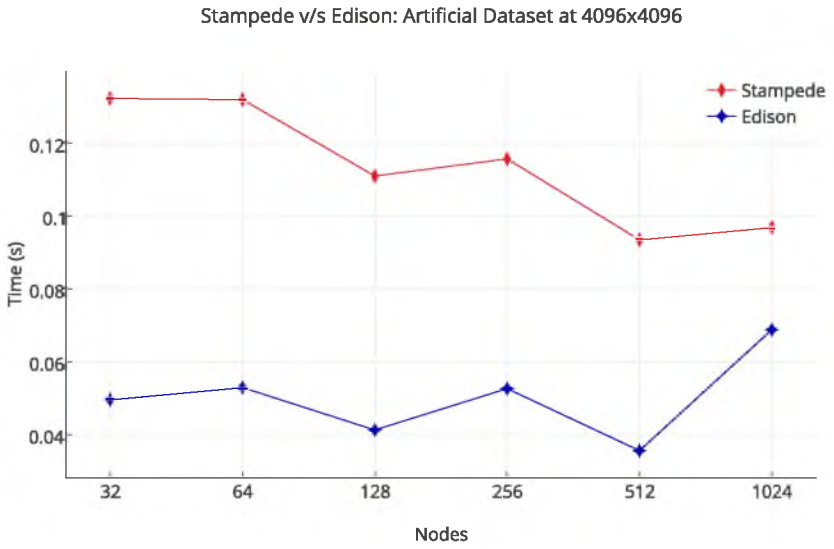

Fig. 4.10. Comparing Stampede and Edison for up to 1024 nodes for the artificial dataset at 4096x4096 resolution.

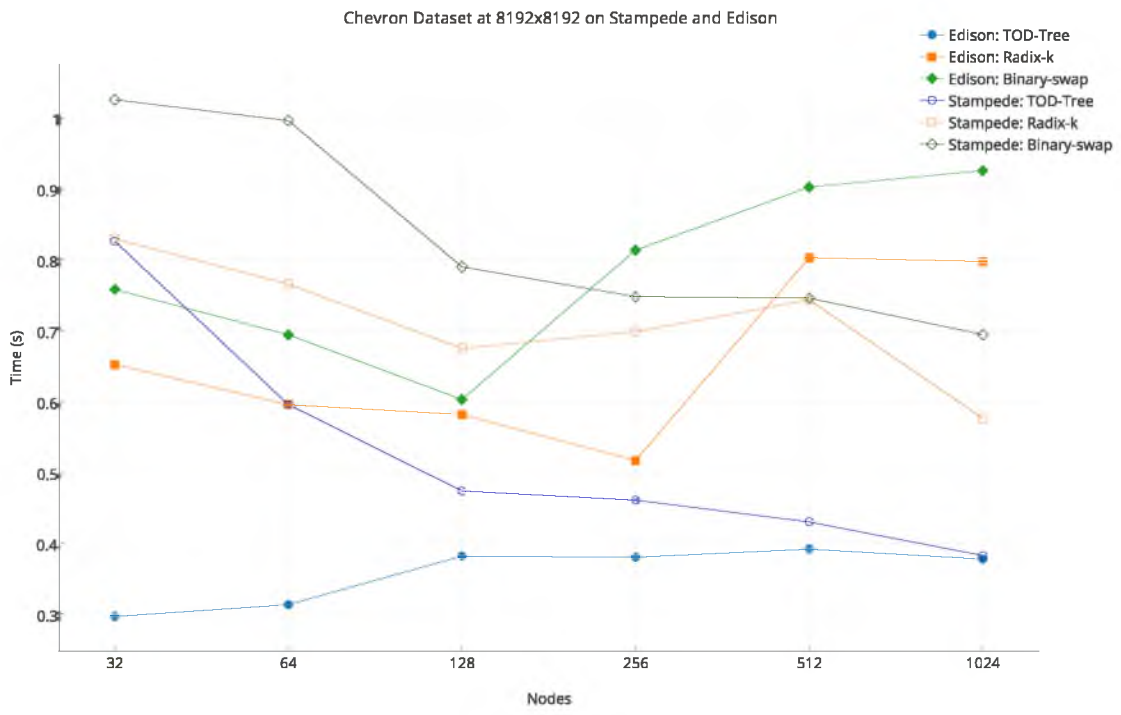

Fig. 4.11. Comparing Stampede and Edison for up to 1024 nodes for combustion at 8192x10418 resolution.

CHAPTER 5

DISTRIBUTED VOLUME RENDERING WITH COMPOSITING ON GPU-ENHANCED SUPERCOMPUTERS

5.1 Introduction

In the push towards exascale, a number of new supercomputers are being deployed in the coming years. Three of these supercomputers, Cori at NERSC, Trinity at the Los Alamos National Laboratory, and Aurora at Argonne National Laboratory, are CPU-accelerated supercomputers and two of them, Summit at Oak Ridge National Laboratory and Sierra at Lawrence Livermore National Laboratory, are GPU-enhanced supercomputers. Therefore, having an algorithm that can work on both CPU-only and GPU-accelerated supercomputers is thus very important. Currently (April 2016), two of the top 10 of the Top 500 supercomputers are equipped with Nvidia GPUs. GPUs have been so successful for General Purpose computing on GPU (GPGPU) that although they were initially developed for accelerating graphics, they are now mostly used in supercomputers for computing rather than for graphics. Until recently, GPUs could be used only for compute or graphics, but not both. However, Nvidia Tesla class GPU K20 and above can run both graphics and compute at the same time. Moreover, whereas inter-node communication between GPUs previously had to go through the CPU, with the introduction of GPU Direct Remote Direct Memory Access (RDMA), GPUs can communicate directly over a network with minimal latency. These two changes allow us to do both rendering and compositing on the GPU since GPUs are at least twice as fast as CPUs for raycast rendering [16].

5.1.1 Main Contribution

This chapter extends the work from the previous chapter where we compared the performance of the TOD-Tree algorithm against radix-k and binary-swap on an artificial and combustion dataset on CPU-enhanced supercomputers. Here, we extend this algorithm to GPU-accelerated supercomputers. The new contributions are:

- development of a multi-GPU compositing algorithm based on TOD-Tree that takes advantage of modern GPU capabilities
- scaling to 4096 GPUs on Piz Daint, a GPU-accelerated supercomputer
- a workflow that allows seamless transfer, with minimal latency, of renderings from an OpenGL context to a CUDA context and uses GPU Direct RDMA for compositing

Whereas volume rendering is often done on GPUs, compositing is usually done on the CPU [55], [79]. In this work, we do both on the GPU. The only image compositing algorithm that we have found for GPUs is parallel direct send in the vl3 system [38], which has been scaled to 128 GPUs on the Tukey computer cluster at Argonne. In this chapter, we scale to 4096 GPUs on the GPU-accelerated supercomputer Piz Daint. As far as we know, this is the largest scaling across GPUs. We compare the performance of TOD-Tree scaled to 4096 nodes on two CRAY XC30 systems: Edison, a CPU-only supercomputer, and Piz Daint, a GPU-enhanced supercomputer. We show that GPU compositing achieves performance on par with CPU compositing for 2K x 2K and 4K x 4K images, and even better performance for 8K x 8K images.

Most visualization software uses OpenGL and shaders to do volume rendering on GPU. However, GPU Direct RDMA, which allows GPUs to talk across a network, does not work in OpenGL; it only works using CUDA. So after rendering in OpenGL, we need to switch over to CUDA for image compositing. Transferring data from OpenGL to CUDA can be easily done using the CUDA OpenGL Interoperability runtime. The usual render target for OpenGL offscreen rendering are textures, which is mapped to CUDA arrays using the CUDA OpenGL Interoperability. CUDA arrays reside in texture memory but GPU Direct RDMA

does not work with texture memory, only device memory. Moving data from texture memory to device memory can be quite expensive, so we instead render to an OpenGL buffer object that can be mapped to device memory. The workflow we introduce shows how to do rendering and image compositing using the GPU and what is required to modify existing systems to do all the visualization on the GPU. This workflow could be very useful for in-situ visualization where simulation and visualization can proceed in parallel on the CPU and GPU, respectively. As far as we know, this is the first time this workflow has been used.

5.2 Methodology

As mentioned before, distributed volume rendering has three stages: loading, rendering, and compositing. Rendering on the GPU is fast, but compositing has usually been done on the CPU because of the high latency of inter-node GPU communication. The workflow that we describe explains how to minimize the latency for inter-node GPU communication.

5.2.1 Workflow for Rendering on the GPU

OpenGL with shaders is the most common option for doing volume rendering on the GPU, but the only technology that allows GPUs to talk across a network is GPU Direct RDMA, which is available only in CUDA. In this section, we describe the workflow that allows the seamless transfer of data rendered from the OpenGL graphics pipeline to CUDA using the CUDA OpenGL interoperability runtime.

All OpenGL programs need an OpenGL context. To create a context on Linux, the operating systems that most HPC systems use, an X server is required. The X server is a program that sits on top of the driver and handles input and output from an application. To create a context, the Xlib library is used to connect to the X server, and GLX is then used to create a context. On desktop systems, an X server is usually started by default, but on compute nodes of HPC systems, the X server might have to be explicitly started using $\#SBATCH --constraint = startx$ in the job submission script to the job scheduling system. In the future, once most GPU drivers in supercomputers have support for EGL [80], we should not have to

initialize an X server to create an OpenGL context. Fig. 5.1 shows the interaction that goes on with the GPU, driver, X server, libraries, and application.

Compute nodes in supercomputers are rarely connected to displays. OpenGL rendering is, therefore, usually offscreen targeted to a framebuffer object or renderbuffer object, both of which are usually mapped to texture memory in OpenGL. When using CUDA OpenGL interoperability, they will be mapped to texture memory in CUDA, but GPU Direct RDMA does not work from texture memory. There are two ways to map data from texture memory to device memory in CUDA. It can be copied to device memory using *cudaMemcpyFromArray* and *cudaMemcpyDeviceToDevice* or through a CUDA kernel. However, in some tests we ran, we found both approaches to be slow for large textures. Using *cudaMemcpyFromArray*, it took about 5 milliseconds for a 4,096 x 4,096 RGBA32F image and about 21 milliseconds for 8,192 x 8,192 RGBA32F image. Therefore, instead of rendering to a framebuffer object, we render to an OpenGL Buffer Object, more specifically to a *GL_TEXTURE_BUFFER*, which is mapped to device memory when using CUDA OpenGL interoperability.

A *GL_TEXTURE_BUFFER* can store up to 134,217,728 million pixels (a maximum image size of 8,192 x 16,384 pixels) and behaves like a regular OpenGL texture but is only one-dimensional. To store the output of a fragment shader to it, we need to map the (x,y) screen coordinates to a one-dimensional position in GLSL as follows:

Listing 5.1. Computing fragment location

```
int index;
index = (int(floor(gl_FragCoord.y)) - minY)*
 width +(int(floor(gl_FragCoord.x)) - minX));
```

where width is the width of the screen, minX and minY are the minimum x and y coordinate, and gl_FragCoord is an OpenGL variable that stores the coordinates of a fragment in screen space.

The steps to render to a *GL_TEXTURE_BUFFER* instead of a framebuffer in OpenGL are:

1. initialize a *GL_TEXTURE_BUFFER* and bind it to a texture

2. pass the texture and its width and height and minimum x and y values to the shader
3. to receive the uniform in the shader:
   $layout(rgba32f, binding = X)\ coherent\ uniform\ \ imageBuffer\ imgOut;$
   (where X is the texture number and imgOut is the name of the texture)
4. compute the index of where to store the fragment as shown in listing 5.1
5. use imageStore to store the fragment.

Once rendering is done, the texture buffer object can be mapped to CUDA device memory using the *cudaGraphicsGLRegisterBuffer* function. Then CUDA kernels are used for blending and GPU Direct RDMA for communication. Rendering to an OpenGL buffer object instead of the usual framebuffer is key in the workflow, shown in Fig. 5.1, since it allows latency to be kept to a minimum by not having to copy any data. Also, changing the rendering target to a texture buffer object in an existing program should be quite straightforward.

5.2.2 Compositing Algorithm

Compositing has two main components: communication and computation. In the previous section, we explained how communication and computation will be done on the GPU. However, we also need the logic for doing compositing; we need an algorithm. The algorithm we used is the TOD-Tree algorithm. Compared to algorithms such as radix-k and binary swap, TOD-Tree tries to minimize communication and hides the latency of inter-node communication by overlapping computation with communication. Since GPUs can blend images faster than CPUs, communication avoidance is even more important on GPUs than on CPUs. Implementing TOD-Tree on the GPU is quite similar to implementing it on the CPU. The only changes needed are blending and memory allocation. For blending, CUDA kernels are used on the GPU instead of OpenMP with vectorization on the CPU, and memory allocations and deallocations are through *cudaMalloc* and *cudaFree*. No change is needed to use GPU Direct RDMA in the program; the same MPI calls are made but the buffers used are in CUDA device memory. In the job script submitted to job scheduling system, *export MPICH_RDMA _ENABLED_CUDA* = 1 is needed

to activate GPU Direct RDMA, which is verified in the program by checking the environment variable using *getenv*("*MPICH_ RDMA_ENABLED_CUDA*").

For the rendering stage on the GPU, OpenGL 4.4 and GLSL shaders were used to implement ray casting volume rendering. The same algorithm was implemented in C++ for the CPU.

5.3 Testing and Results

5.3.1 Test Setup

Most supercomputers have CPUs with many cores, and some are also enhanced by coprocessors such as Nvidia Tesla GPUs, which have thousands of cores. For this project, we ran our algorithm on a GPU-enhanced supercomputer and compared our results against a CPU-enhanced supercompter both CRAY XC30 systems. Also, since IceT was not built to run on GPU, we could not do a performance comparison using IceT on Piz Daint. Instead, we compared the performance of TOD-Tree between CPU and GPU compositing on Edison and Piz Daint, since both are CRAY XC30 systems. The test dataset used for comparison is the artifical box dataset described in the previous chapter.

The primary test system used for testing is Piz Daint, a Cray X30 supercomputer that uses the dragonfly topology for its Aries interconnect network. The 5,272 nodes are arranged into 28 cabinets. Each node has an Intel SandyBridge processor with eight cores (Intel Xeon E5-2670) that has a peak performance of 211 GFLOPS and a Nvidia Tesla K20X GPU that has a peak performance of 3.95 TFLOPS [81]. The peak performance of Piz Daint is 7.787 PFLOPS [82]. On Piz Daint, we ran TOD-Tree on the GPU using GPU Direct RDMA for communication and CUDA kernels for computation. The system we compared against was the Edison supercomputer at NERSC. Edison is a Cray X30 supercomputer that uses the dragonfly topology for its Aries interconnect network. The 5,576 nodes are arranged into 30 cabinets. Each node is an Intel IvyBridge processor with 12 cores (Intel Xeon E5-2695v2) and has a peak performance of 460.8 GFLOPS/node. The peak performance of Edison is 2.57 PFLOPS [76].

For a fair comparison between between CPU and GPU, all the parallelism aspects of the CPU must be used. The TOD-Tree algorithm on Edison has been compiled using autovectorization, which uses SIMD parallelism, and OpenMP, which uses all the cores of the CPU, on Edison. On Piz Daint, we ran TOD-Tree on the GPU using GPU Direct RDMA for communication and CUDA kernels for computation. The same value of r for pairs of time steps; r=16 for 32 and 64 nodes, r=32 for 128 and 256 and, r=64 for 512 and 1024, r=128 for 2048 and 4096, and only 1 round was used for the k-ary tree part of the algorithm. The GPU on Piz Daint is much more powerful than the CPU on Edison: the Tesla K20X on Piz Daint has a peak performance of 3.95 TFLOPS compared to the 460.8 GFLOPS on Edison.

5.3.2 Scaling on Piz Daint

On Piz Daint, we had access to 3,000 node hours, which did not allow us to run as many tests as on the other platforms, but we still managed to scale up to 4096 nodes/GPUs using the TOD-Tree algorithm for 2048x2048, 4096x4096 and 8192x8192 images for the artificial dataset. The results are shown in Fig. 5.2.

The 2048x2048 image, topmost graph in Fig. 5.2, has numerous fluctuations. These fluctuations, however, all take place within 6 milliseconds, meaning that they will barely affect the rendering frame rate. For 2048x2048 images, the overall size of the full image is only 64MB, and the many variations can be explained by the fact that performance is mainly communication bound. These fluctuations decrease as the size of the image increases, and the compositing starts to be more computation bound than communication bound. The average coefficient of variation for compositing time is 10.3% for 2048x2048 images, 3.7% for 4096x4096 images, and 1.8% for 8192x8192 images. The sawtooth appearance is similar to what we see on Edison and Stampede since the same values are used for the parameters r and k for the same number of MPI processes on all three systems.

We compared running TOD-Tree on Edison with Piz Daint since we ran with the same number of MPI processes on each, and both Edison and Piz Daint are CRAY XC30 systems with the same dragonfly topology and Aries interconnect network. The compositing times are very close for 2048x2048 and 4096x4096. The

difference in time is within 5 milliseconds for 2048x2048 images and usually within 10 milliseconds for 4096x4096 images with a maximum variation of 20 milliseconds at 1024 nodes. For the 8192x8192 image, TOD-Tree is much faster on Piz Daint because we believe that for 8192x8192 image, compositing is more computation bound and computation on Piz Daint is faster than on Edison. If we compare the increase in average compositing time for the 2048x2048 to 4096x4096 image (for which the size increases by 4), we see that it has increased by, on average, 3.7 times on Edison and 3.2 times on Piz Daint. For 4096x4096 to 8192x8192, the average increase in compositing time is 7.2 on Edison compared to 3.7 on Piz Daint, again for a size increase of a factor of 4. The increase in time on the GPU is quite consistent, as shown in Fig. 5.3.

5.3.3 Scaling Across Machines

Fig. 5.4 shows the result of TOD-Tree algorithm on Stampede, Edison, and Piz Daint. The values of r and k used are the same on all three supercomputers. As expected, the algorithm is faster on Edison and Piz Daint compared to Stampede: the Aries interconnect on the CRAY XC30 is faster and the nodes have better peak FLOP performance. Whereas on Stampede, we are not using threads, on Edison we are using threads and vectorization and using CUDA kernels on Piz Daint. The gap between the performance is larger for low node counts, as each node has a bigger chunk of the image to process when few nodes are involved, a faster processor makes quite a big difference. As the number of nodes increases, the data to process decreases and so the difference in computing power is less important as the compositing becomes communication bound. The sawtooth appearance is present on all three systems. On average, we are still getting about 16 frames per second for a 256MB images (4096x4096 pixels). At 2048 nodes, the time taken for TOD-Tree decreases, as can be seen in the middle chart of Fig. 5.2.

Finally, if we look at how the performance of TOD-Tree varies as we increase the number of nodes from 32 to 4096 nodes for the 2048x2048, 4096x4096, and 8192x8192 images, we notice that there are more spikes and troughs in the 2048x2048 and 4096x4096 sized images than in the 8192x8192 sized image. The main reason for

these many fluctuations is that image compositing is communication bound for small images. When doing image compositing for 2048x2048 sized images, the data exchanged among nodes is usually quite small and so the ratio of latency to interconnect transfer speed is quite high, resulting in an overall bandwidth low. For large images, the ratio of latency to interconnect transfer speed is low and so we benefit from higher bandwidth speed, which makes image compositing less communication bound. Communication-bound processes tend to have more variations due to the unpredictability of network traffic. These variations are more visible in the 2048x2048 sized image since the small fluctuations in network speed are more visible when we are looking at resolutions of milliseconds compared to resolutions of 50 or 100 milliseconds as in Fig. 5.4.

5.4 Summary

TOD-Tree performs equally well on GPU-accelerated supercomputers, which are even better for large images due to the higher peak performance of GPUs. There is a large difference between the computational power available to one node compared to the speed of inter-node communication. Computation is usually at least one order of magnitude faster than communication, so algorithms must be designed to pay much more attention to communication than to computation if we are to achieve better performance at scale. Also, we have introduced a workflow that enables us to seamlessly transfer data from OpenGL to CUDA to allow faster overall rendering that can be easily integrated with existing GPU volume rendering systems.

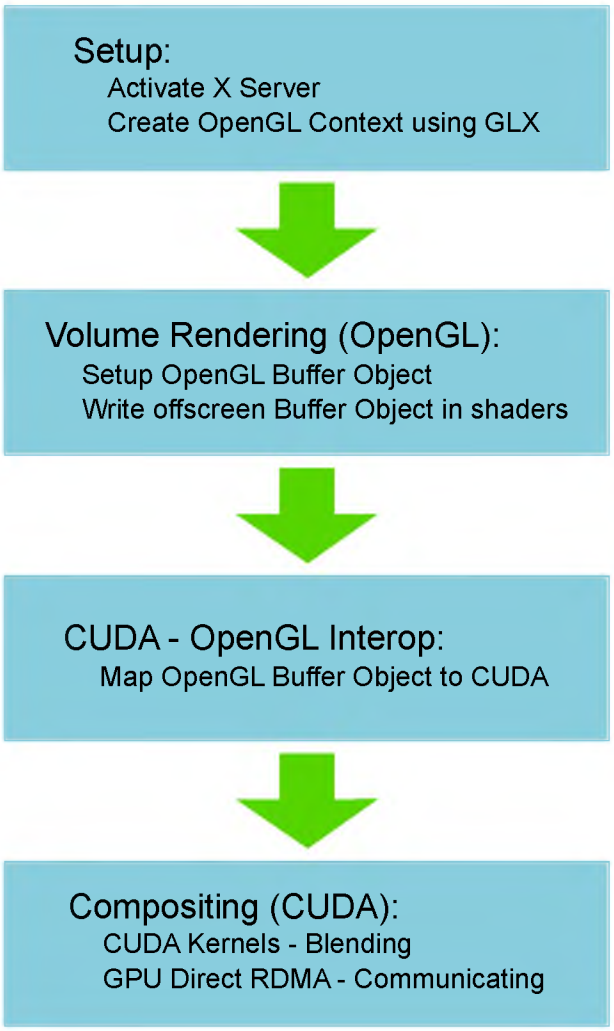

Fig. 5.1. Workflow for GPU rendering.

Fig. 5.2. Comparing scaling for Edison and Piz Daint.

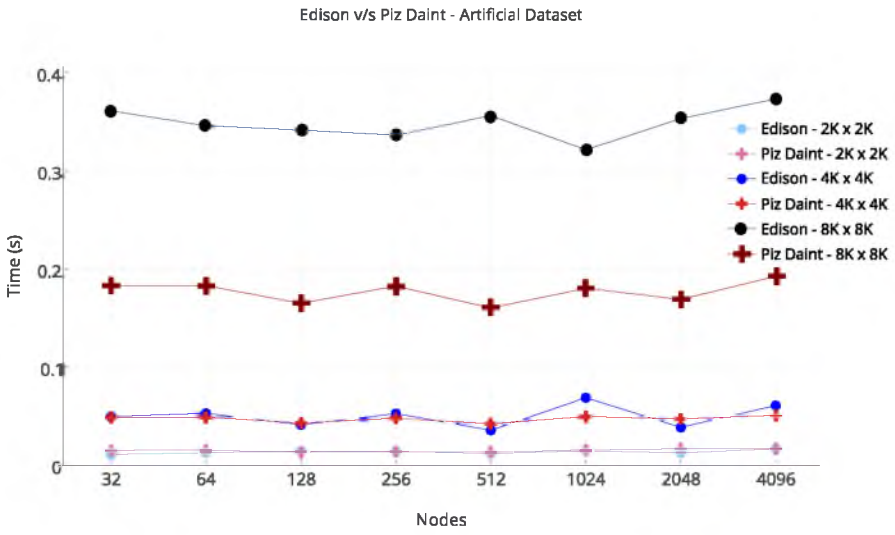

Fig. 5.3. Comparing scaling on Edison and Piz Daint for 4096 MPI processes.

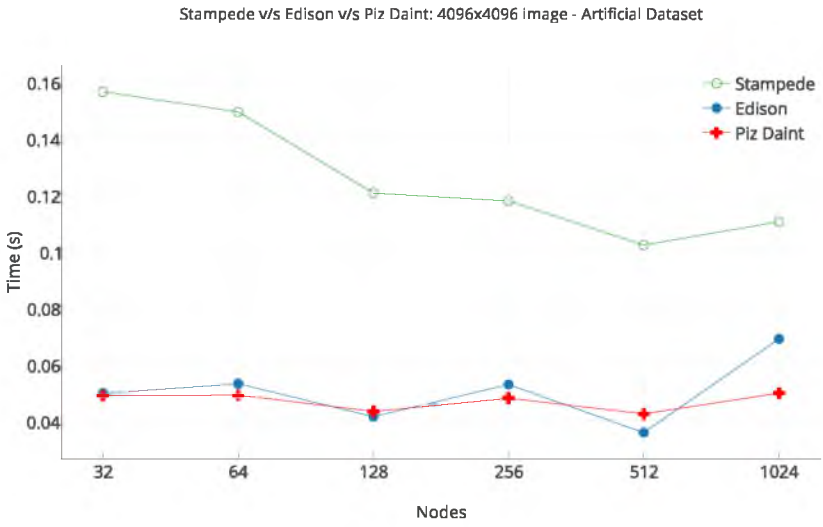

Fig. 5.4. Comparing Stampede and Edison for up to 1024 nodes for the artificial dataset at 4096x4096 resolution.

CHAPTER 6

DYNAMICALLY SCHEDULED REGION-BASED IMAGE COMPOSITING

6.1 Introduction

Sort-last distributed volume rendering has three stages: loading, rendering, and compositing. The data to be visualized is typically divided so that each node has the same amount of data, and ideally, loading and rendering should take the same amount of time on each node of a distributed memory machine. However, the rendering time is rarely the same across nodes. There are three main reasons for this: (1), the features that users want to see are rarely uniformly distributed, an example is shown in Fig. 6.1, and the nodes assigned to rendering these empty regions, made invisible through a transfer function, have much less work to do and will finish early; (2), when using perspective projection, nodes closer to the camera produce a larger image compared to nodes far from the camera; (3), if the user zooms in on one specific region of a dataset, part of the dataset might fall outside the viewing window and not need to be rendered at all.

6.1.1 Main Contribution

The main contribution of this work is an image compositing algorithm that uses a scheduler with both spatial and temporal awareness of the compositing process. We start by dividing the final image into a number of regions r and create a depth-ordered list of nodes for each region. Based on the data loaded by each node and the properties of the camera, the spatial contribution of each node to regions of the final image can be determined. Nodes not contributing to a region can then be removed from that region's list. The scheduler also updates the region list after each node is done rendering by eliminating nodes that rendered nothing for a

region. This process ensures that a node not contributing to a region is never made to receive data for that region, thus minimizing communication. The algorithm then schedules the exchange of images and ensures that no nodes wait for a node that is still rendering if another option for compositing is available. Thus when the slowest node is done rendering, most of the regions of the final image have already been composited and there is minimal overhead to assemble the final image. The algorithm uses one MPI rank per node and threads for CPU cores, which Howison et al. [42] showed to be better than one MPI rank per core. Auto-vectorization is also used to fully leverage the compute capabilities of modern CPUs, and asynchronous MPI communication is also used to overlap communication with computation. We compare this scheduling-based image compositing algorithm against TOD-Tree on the Edison supercomputer at NERSC using a box and sphere artificial dataset and a combustion dataset.

6.2 Methodology

As mentioned before, it is rare for rendering on all the nodes of a distributed memory machine to finish at the same time. Improving compositing time, therefore, requires minimizing the time between when the slowest process finishes rendering and compositing is complete; the orange region in Fig. 6.2. For that to happen, processes still rendering should not delay compositing.

One of the issues with compositing algorithms such as parallel Direct Send, Binary Swap, Radix-k, and TOD-Tree is their lack of awareness of which processes have finished rendering and which processes are still rendering, which sometimes delays image compositing as processes wait for images from processes that are still rendering. Fig. 6.3 shows an example of eight processes doing compositing using radix-k. Let us assume that two rounds are needed and vector $k = 4,2$. If processes 6 and 0 are still rendering while the remaining processes are compositing, radix-k will be stuck in round 1 of parallel direct send. A similar delay would occur in Binary Swap and TOD-Tree if some nodes are waiting to exchange images with nodes that are still rendering.

The same set of processes can be represented as a graph, as shown in Fig. 6.4. If we blend exclusively based on depth, processes 4, 1, 7, and 5 can start compositing while waiting for 6 and 0 to finish rendering. Also, since there are never any cycles in the graph, we will refer to it as a chain.

This procedure, however, can still be improved upon. If 6 and 0 do not contain information relevant to the whole image, they should not delay compositing for the whole image. We can then split the image into several regions and have a chain for each region of the image. Fig. 6.5 shows an image split into four regions with a depth-sorted chain for each region. As we can see, each chain has a different length since a node will rarely contribute to all regions of an image. As the number of processes increases to hundreds or even thousands, the contribution of one process to the whole image lessens. Therefore, stalling the whole compositing because of a few slow rendering processes can be avoided; we need to stall only a few regions. The key here is the spatial awareness that is inherent in our proposed algorithm.

For our algorithm, we divide the image into a set of r regions with a depth-sorted chain for each region. To create the chains for each region, we can gather information about the extents of the data each process is loading from MPI, or if a k-d tree is used to load the data, this information can be obtained programmatically for each region from the k-d tree. Using the extents and camera information camera, we can compute the depth of each process and the position and area contributed by each process in the final image. For each chain, we also need to decide which processes will be responsible for gathering information. To try to ensure that different nodes are used to collect information for each chain, the first collector node in the chain for region i is the i^{th} node in the chain. The second is the $(i+r)th$ node. If a chain has fewer than r nodes, the last node is made the collector node for that region. The collector processes are marked with a black circle inside in Fig. 6.5. The number of regions in this case is four. The first chain, chain 0 colored pink, has only three nodes, so the last node is set as the collector. The second chain, chain 1 colored cyan, has seven nodes. Therefore, node 1 and node 5 are set as collectors.

6.2.1 Compositing Algorithm

For our algorithm, we have set aside one process that is not involved in compositing or rendering to act as a scheduler. The scheduler builds a chain for each region, and the compositing processes contact the scheduler to determine with which processes they should exchange images. The chain for each region is constructed as indicated in algorithm 4.

Algorithm 4: Initialize Scheduler

```
Collect the depth and extents for each process
Sort the processes based on depth
Construct a chain based on depth
for each region do
    Use the computed depth chain as the starting point
    Compute and store the extents for that region
    for each process in the chain do
        Compute the extents of the process
        if extents of process does not overlap the chain's then
            Delete process from the chain
            Adjust the to and from neighbor for deleted process
    if length of chain ¡ number of regions r then
        Set last process as collector
    else
        Set every r process to be a collector
Create a buffer for final receive
Launch asynchronous MPI receive for final image
```

Based on the depth information from each process, a depth-sorted chain, as shown in Fig. 6.4, is constructed and is used as the initial chain for each region. For each region, processes that do not contribute to that region are removed from the chain, which creates spatial awareness for each region and reduces the length of each chain. In software, each chain is implemented using a hash map, unordered_map in C++, so that access time is always $O(1)$, and each node of the chain stores the neighbor to and from it. The last step is to create a buffer to receive the composited image for each region. This step ensures that when the final image regions are sent to the display node, they are not written to a temporary buffer but directly to the final image.

The scheduler is then started and awaits communication from compositing processes. Algorithm 5 shows the algorithm for the scheduler. If the scheduler is receiving information from a process for the first time, it also receives the extents of the rendered image. The chain for each region was initialized based on the expected rendered extents from each process, but depending on the transfer function, some regions might not have been rendered for a process. Based on the rendered extents, therefore, some nodes are removed from region chains if they do not have any information for that region. If that process p was marked as a collector process for a specific region, its neighbor is made a collector process, and the process p is deleted to minimize unnecessary transfer of data to that process.

Next, the scheduler performs dynamic scheduling by deciding which processes should communicate with each other. In each region for which the received process is active, the received node in that chain is marked as ready, and the chain is checked to see if there is any neighbor process marked as ready. If a valid neighbor is found and one of them is a collector process, the noncollector will send its data to the collector process. Otherwise, the node having the smaller image will send data to the node with the larger image to minimize communication time. In each case, the sender node is marked for deletion and the receiver is marked as unavailable. The last step of the algorithm is to check if there are any chains that are now empty or have only one remaining ready process. If there is only one ready process, it is made to send its information to the root node and the chain is cleared. The next step is to send all the information at once to each node that has work to do. A process might need to send data to a node x for a specific region and receive data from the same node x for another region. All the communication to a node from the scheduler is done in one step.

Each compositing node runs the compositor algorithm shown in algorithm 6. The first time a process communicates with the scheduler once it is done rendering, it sends its extents to the scheduler. As mentioned before, based on the transfer function, a process will not always render all data it has loaded, and as spatial awareness is a key component of our algorithm, we want to update the region chains to reflect the state of the rendering. Also, each process will receive in one

Algorithm 5: Scheduler

```
while !done do
    Wait for communication from the rendering process
    if first communication from the process then
        Receive rendered extents from the process
        for each region do
            Determine extents for region
            if extents of a process does not overlap the chain's then
                Remove the process from the chain
                Adjust neighbors to and from for deleted process
    Mark node as active in a chain where it exists
    for each active chain do
        if only process in chain then
            Mark a process to send information to root
            Erase the chain
        else
        Find neighbor for incoming node
        if neighbor found then
            Determine if sender or receiver
            Mark receiver as processing
            Delete sender from chain
            Save sender and receiver information
    for each active chain do
        if size is 1 AND process is ready then
            Process will send data to root
    for each process marked for communication do
        Send information
    if all chains are empty then
        Exit Scheduler
```

message all the other processes with which it needs to communicate to keep communication in the system to a minimum. Information for each communication will contain the neighbor with which to communicate, the region, blending direction, and MPI tag. Also, each send from a process is in the form of an asynchronous send to maximize overlapping communication with computation.

6.2.2 Choosing Number of Regions

For the scaling run, we have set the number of regions to be 16. This number was determined after a series of initial test runs in which we experimented with

Algorithm 6: Compositor

```
Get the extents of the image rendered by the process
Count the number of active regions (countActiveRegions) covered by the
image
while !done do
    if first time then
        Send extents to the scheduler
    else
        Tell scheduler that it is ready
    Wait for scheduler to respond
    for each process to communicate with do
        if Only one process in chain then
            Send data to root
            countActiveRegions -= 1
        else
            if Send then
                Async send to neighbor
                countActiveRegions -= 1
            else
                Receive image
                if last round then
                    Create opaque image
                    Create alpha buffer
                    Blend current image with the background
                    Blend in opaque buffer
                    Send to display node
                    countActiveRegions -= 1
                else
                    Blend with image on node
    if no active regions left then
        Exit loop
```

1, 2, 4, 8, 16, and 32 regions for 4,096 x 4,096 sized images. When few regions are used, a slow node impacts few regions, but since each region occupies a substantial portion of the image, compositing ends up being slow. For example, if we use only two regions for an 8K x 8K image, and there is one slow node in the upper region, half of the compositing is delayed by one node. If too many regions are used, one slow node will impact many small regions. Since there are many regions, the overall impact of a slow region will be less. However, this will result in a lot of

communication with many exchanges, which we want to avoid. Sixteen regions provided a right balance between avoiding too much communication and one node having too much of an impact on the whole compositing process.

6.3 Testing and Results

We first examine the setup for the experiment, the test platform, data, and algorithm used before presenting the scaling results.

6.3.1 Experiment Setup

The test platform used is the Edison Cray XC30 supercomputer at NERSC. Edison uses the Cray Aries high-speed interconnect with Dragonfly topology that has an MPI bandwidth of about 8 GB/sec and latency in the range of 0.25 to 3.7 usec. It has 5,576 compute nodes, each of which has two 2.4 GHz 12-core Ivy Bridge processors with 64 GB of memory per node. We scaled up to 2,048 nodes of the 5,576 nodes of Edison.

The test datasets that we used are a box and sphere artificial test datasets and a combustion dataset shown in Fig. 6.6. The combustion dataset has 5,996 blocks of scalar data, each of which has about 17,000 cells per block. At the bottom of the dataset, there are a number of tubes through which fuel is injected into the combustion chamber. Combustion starts above these tubes and rises to the top of the combustion chamber, hitting the ceiling and the walls. When visualizing this dataset, much more work has to be done in the upper regions of the dataset, thereby creating an imbalance in the rendering workload. The artificial datasets are simpler: each rendering process is assigned one block of uniform scalar data per node. The box dataset is similar to what was used by Moreland et al. [83], and we also introduced a sphere dataset whose diameter is equal to the length of the cube.

The algorithm we compared against is the TOD-Tree algorithm described in Chapter 4 that has been shown to perform generally better than Radix-k. Both TOD-Tree and our algorithm use threads and auto-vectorization compared to the ICET library [15], which does not use threads.

6.3.2 Scheduler Cost

Building and running the scheduler is fast: the time it took to construct the region chains and for MPI Gather to collect the depth and extents information from each node, for 2,048 nodes, was measured to be on average 0.5 milliseconds. The time it took the scheduler to respond to a compositing node if neighbors were available was on average 0.2 milliseconds. With a latency of at most 3.7 microseconds, the cost of communicating with the scheduler is minimal compared to the cost of exchanging data among nodes.

6.3.3 Scaling Studies

For each of the three datasets, and for each of the three image sizes used (2,048 x 2,048, 4,096 x 4,096, and 8,192 x 8,192 pixels), we performed 10 runs after an initial warm-up run, and the results are the average of these runs after some outliers have been eliminated.

Fig. 6.7 shows the total time it takes to render and composite the combustion dataset for up to 2048 nodes on Edison. As expected, as the number of nodes increases, the total time it takes to render the dataset decreases. The focus here is image compositing and so, for the remainder of this section, we focus on compositing.

Depending on the amount of rendering work each node has to do, compositing will start at different times on each node. The compositing time that needs to be minimized is the time interval between the slowest rendering job finishing and the final image is ready on the display node: the orange region in Fig. 6.2. Any compositing done in the interval of time between the fastest rendering node and the slowest rendering node does not slow down the entire compositing process. The compositing time that we measured and plotted in Fig. 6.8 and 6.9 is the time interval between the slowest rendering job and the image being ready on the display node. Our dynamically scheduled region-based compositing algorithm is labeled as DSRB in the figures.

Fig. 6.8 shows the compositing time for the combustion dataset for 2,048 x 2,048 (2Kx2K), 4,096 x 4,096 (4Kx4K), and 8,192 x 8,192 (8Kx8K) sized images. When

there are few nodes, each node renders a larger region and so influences many regions of the chain. Therefore, we do not gain much from overlapping rendering with compositing since compositing in most regions is stalled by waiting for other nodes. As the number of nodes increases and the contribution of each node to regions is decreased, the overlapping of compositing and rendering allows us to perform better than the TOD-Tree, which does not have any spatial or temporal awareness of the image being rendered from each node. We also see that there is more variation for the 2K x 2K image compared to the 4K x 4K image and 8K x 8K image since it is more communication bound. The 8K x 8K image has the least variation as it is more compute bound.

The Dynamically Scheduled Region-Based compositing algorithm also performs faster than the TOD-Tree on the artificial dataset. The difference in compositing times between the sphere and box is minimal in most cases. However, since there is less data for the sphere dataset, it takes less time to render compared to the box dataset; the orange box in Fig. 6.2 is smaller. The result of that is that the box seems to have a faster compositing time since what we are showing as compositing time is the time interval between the slowest rendering and the final image being ready. For the TOD-Tree, the sphere is generally faster since there is overall less data to process. As with the combustion dataset, compositing gets faster as the number of nodes increases. Here again, when more nodes are used, each node has a smaller share of the entire image, and a slow node impacts fewer regions, resulting in faster compositing.

6.4 Summary

In this work, we have introduced an image compositing algorithm that has both spatial and temporal awareness of compositing. Spatial awareness ensures that no compositing processes will ever receive data for a region to which it does not contribute, thereby minimizing communication. Temporal awareness ensures that processes do not try to communicate with processes that are still rendering, thereby minimizing delays. Combining spatial and temporal awareness streamlines compositing by allowing several regions of an image to be fully composited

fairly quickly. Compositing is delayed only for data-intensive regions of an image, which gives us a substantial gain compared to TOD-Tree, which lacks spatial and temporal awareness.

Fig. 6.1. Two commonly used test datasets: the Bonsai dataset on the left and Backpack dataset on the right with numerous empty regions in each dataset.

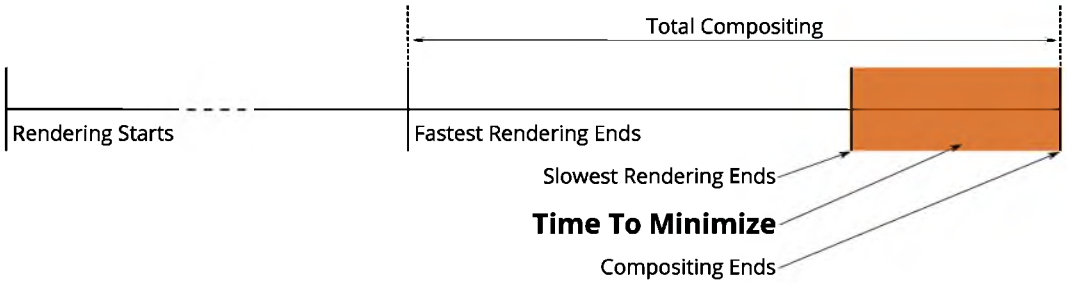

Fig. 6.2. Rendering and compositing timeline.

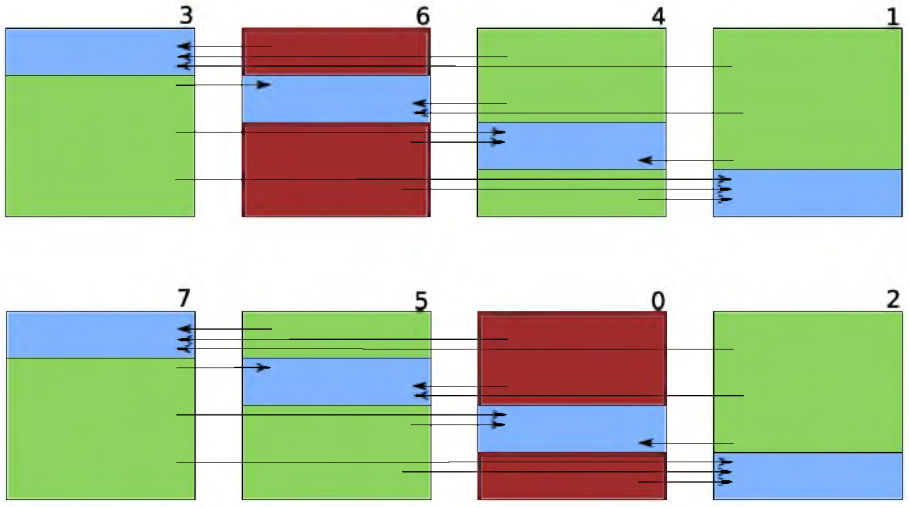

Fig. 6.3. The first round of Radix-k for eight processes. The processes in green are done with rendering and are compositing. The processes in red are still rendering. The blue rectangle shows the region for which each node is responsible.

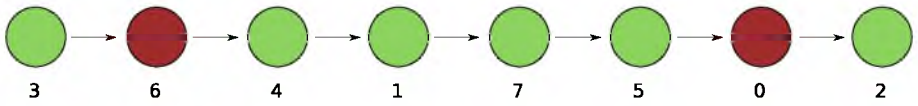

Fig. 6.4. Nodes sorted by depth in a chain.

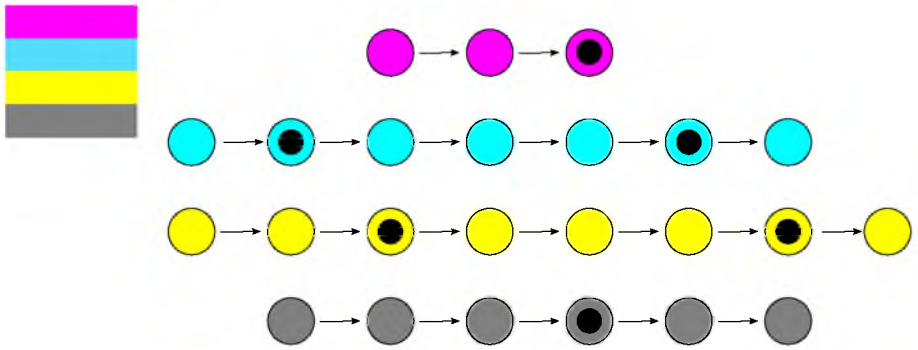

Fig. 6.5. Four chains, one for each of the four regions (purple, blue, yellow, and gray) into which the final image is split.

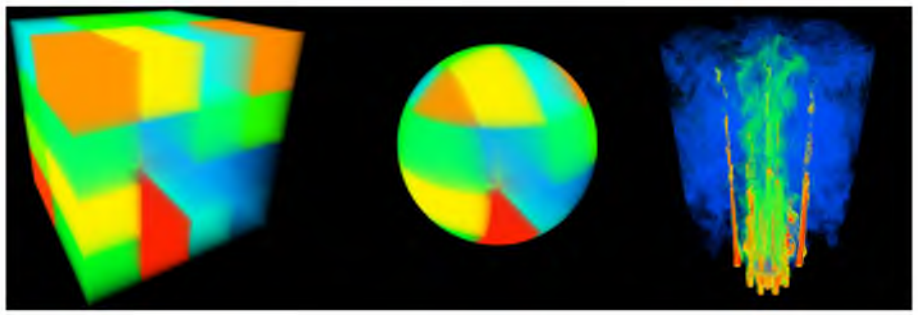

Fig. 6.6. The datasets: box (left), sphere (middle), and combustion (right).

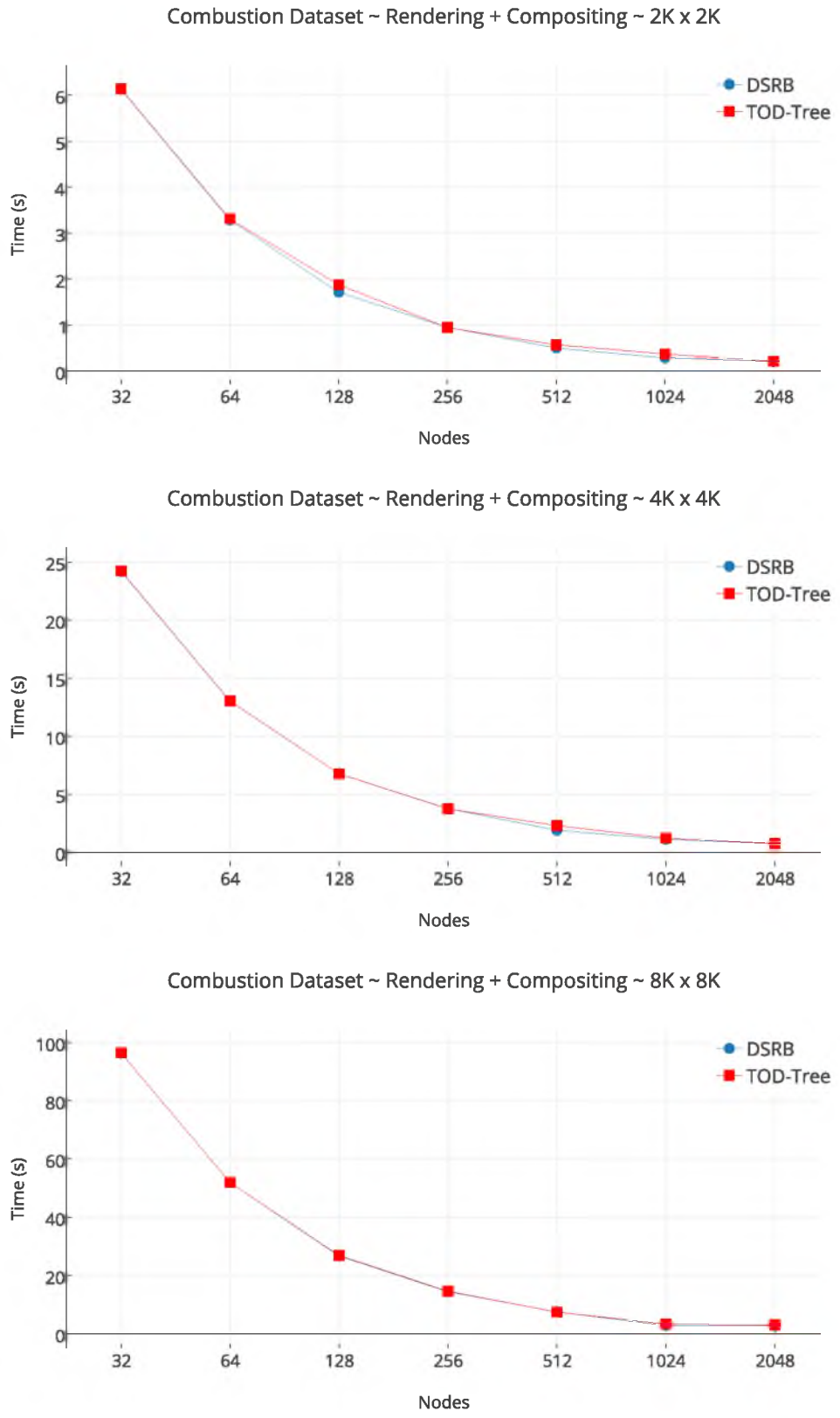

Fig. 6.7. Scaling of the combustion dataset on Edison - showing rendering and compositing.

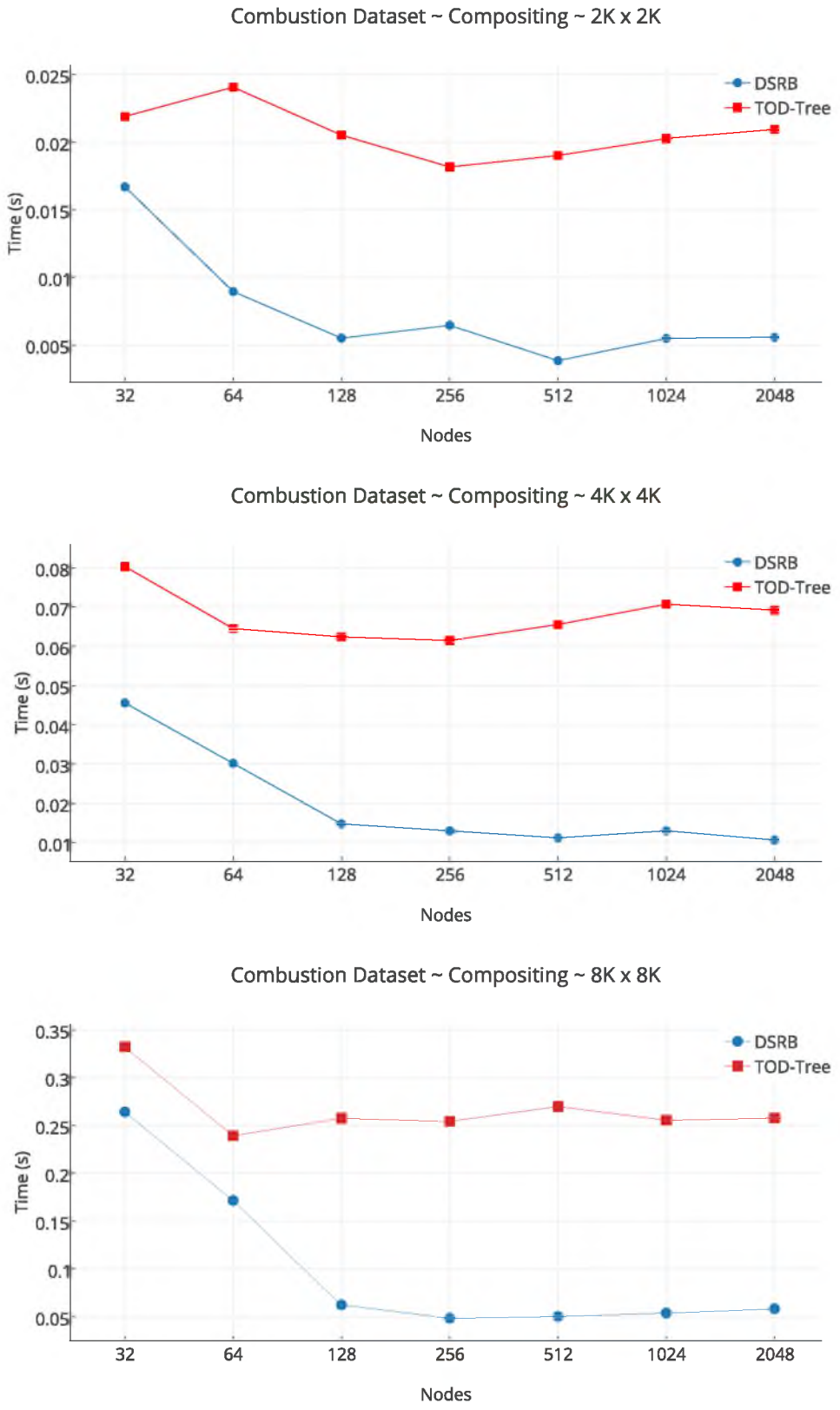

Fig. 6.8. Scaling of the combustion dataset on Edison - showing compositing only.

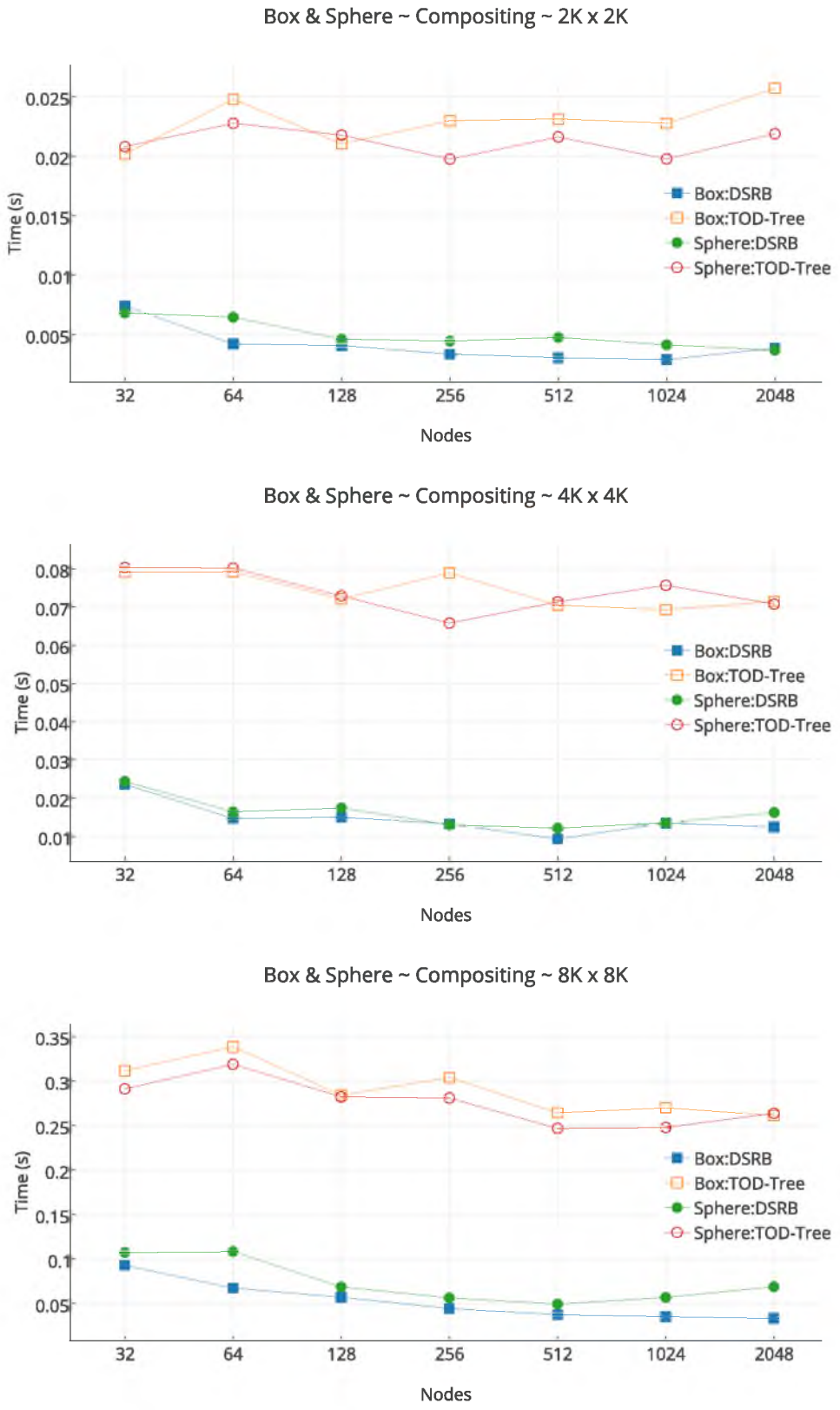

Fig. 6.9. Scaling of the artificial box and sphere datasets on Edison - showing compositing only.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this dissertation, we have looked at two aspects of volume rendering: evaluating how depth of field can be used to improve depth perception and investigating how to make image compositing faster for distributed volume rendering on CPU-only and GPU-enhanced supercomputers.

Direct volume rendering is now commonly used for visualizing data, and many techniques claiming to enhance the quality of the rendering have been developed. However, very few of these techniques have been evaluated to assess their usefulness in improving perception. For the first contribution of this dissertation, we evaluated how DoF impacts depth perception in DVR. We expected to see an overall improvement in all cases, but we found that whereas depth of field on the front features improves depth perception, DoF on the back feature has the opposite effect. Because of depth cue conflict, DoF on the back feature confused the test subjects and negatively affected depth perception.

The second contribution of this dissertation focused on improving image compositing in distributed volume rendering. Sort-last rendering on distributed memory machines is not new, but recent changes in supercomputing architecture have changed the constraints under which compositing algorithms should be built. Previously, the main constraint was to evenly balance the workload among nodes, but the main constraint now is to try to minimize communication and overlap communication with computation. In this dissertation, TOD-Tree image compositing has been developed to minimize communication and overlap communication with computation. Running tests on up to 4,096 nodes on the Edison supercomputer showed that this algorithm is generally faster than binary swap and radix-k. Moreover, using new developments that facilitate inter-GPU communication - GPU Direct RDMA, CUDA OpenGL Interop, and the ability of Tesla class GPUs to run

both compute and graphics - we have developed a pipeline that minimizes latency for distributed volume rendering on GPU-enhanced supercomputers, allowing us to do rendering and compositing exclusively on GPUs. We performed strong scaling studies, on up to 4,096 GPUs of the Piz-Daint supercomputer, to confirm that image compositing on GPU-enhanced supercomputers is at least as fast, and in some cases even faster, than image compositing on the Edison CPU-only supercomputer. Furthermore, we noticed that one of the weaknesses of image compositing algorithms such as binary swap, radix-k, and TOD-Tree is that they do not take into account the distribution of data on each node and which nodes are done rendering and which nodes are still rendering when trying to exchange information for compositing. This results in some nodes waiting for other nodes that are still rendering instead of exchanging images with nodes that are ready to do compositing. Moreover, in some cases, a node is made to be authoritative on a section of the final image for which it has no data, needlessly increasing communication. We therefore developed an algorithm with spatial and temporal awareness that we tested against TOD-Tree and showed that a dynamically scheduled algorithm can be much faster than an algorithm with no awareness of the rendering and compositing state of the rendering nodes.

As future work, we would like to expand the number of test volumes that were used for the depth of field user study. For example, will DoF on the front feature still be able to improve perception in the case of microscopy images? More tests need to be carried out for that. Also, we would like to create an algorithm, based on TOD-Tree, which incorporates the spatial and temporal awareness found in the dynamically scheduled image compositing algorithm. Such an algorithm would allow us to use TOD-Tree when the data is evenly distributed, and leverage spatial and temporal awareness for uneven data distribution, thereby creating a general solution for image compositing for different-sized images and clusters. Moreover, the scheduler is now being run on a different node, but we would like to run it on one of the compositing nodes and finally, we would like to extend our test to larger distributed memory systems for both GPU- and CPU-enhanced supercomputers.

APPENDIX

PUBLICATIONS

- A.V. Pascal Grosset, Peihong Zhu, Shusen Liu, Suresh Venkatasubramanian, Mary Hall. *Evaluating Graph Coloring on GPUs*, ACM SIGPLAN Principles and Practice of Parallel Programming (PPoPP), Feb. 2011, San Antonio, Texas, USA PPoPP'11, February 12-16, 2011 (Runner up for PPoPP 2011 Best Student Poster Award) **(Poster)**
- M. Schott, A.V.P. Grosset, T. Martin, V. Pegoraro, S.T. Smith, C.D. Hansen. *Depth of Field Effects for Interactive Direct Volume Rendering*, In Proceedings of Eurographics/IEEE Symposium on Visualization 2011, Vol. 30, No. 3, Edited by H. Hauser, H. Pfister, and J. J. van Wijk, 2011
- M. Schott, T. Martin, A.V.P. Grosset, C. Brownlee, T. Hollt, B. P. Brown, S.T. Smith, C. D. Hansen. *Combined Surface and Volumetric Occlusion Shading*, Visualization Symposium (PacificVis), 2012 IEEE Pacific, Songdo, 2012, pp. 169-176.
- A.V. Pascal Grosset, M. Schott, G-P Bonneau, C.D. Hansen. *Evaluation of Depth of Field for Depth Perception in DVR*, Visualization Symposium (PacificVis), 2013 IEEE Pacific, Sydney, NSW, 2013, pp. 81-88.
- M. Schott, T. Martin, A.V.P. Grosset, S.T. Smith, C.D. Hansen. *Ambient Occlusion Effects for Combined Volumes and Tubular Geometry* , In IEEE Transactions on Visualization and Computer Graphics (TVCG), Vol. 19, No. 6 (Selected as Spotlight paper for June 2013 issue, pp. 913–926. 2013)
- A.V. Pascal Grosset, M. Prasad, C. Christensen, A. Knoll, C.D. Hansen *TOD-Tree: Task-Overlapped Direct send Tree Image Compositing for Hybrid MPI Parallelism*, In Proceedings of the 15th Eurographics Symposium on Parallel

Graphics and Visualization (PGV '15). Eurographics Association, Aire-la-Ville, Switzerland, 67-76. (Honorable Mention)

- A.V. Pascal Grosset, M. Prasad, C. Christensen, A. Knoll, C.D. Hansen *TOD-Tree: Task-Overlapped Direct send Tree Image Compositing for Hybrid MPI Parallelism and GPUs*, In IEEE Transactions on Visualization and Computer Graphics, IEEE Early Access
- A.V. Pascal Grosset, A. Knoll, C.D. Hansen *Dynamically Scheduled Region-Based Image Compositing*, In Eurographics Symposium on Parallel Graphics and Visualizatio(2016), Gobbetti E., Bethel W., (Eds.), The Eurographics Association.

REFERENCES

[1] Mathias Schott, A. V. Pascal Grosset, Tobias Martin, Vincent Pegoraro, Sean T. Smith, and Charles D. Hansen, "Depth of Field Effects for Interactive Direct Volume Rendering," in *Proceedings of the 13th Eurographics / IEEE - VGTC Conference on Visualization*, Aire-la-Ville, Switzerland, Switzerland, 2011, EuroVis'11, pp. 941–950, Eurographics Association.

[2] M. Levoy, "Display of surfaces from volume data," *Computer Graphics and Applications, IEEE*, vol. 8, no. 3, pp. 29–37, May 1988.

[3] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan, "Volume Rendering," in *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 1988, SIGGRAPH '88, pp. 65–74, ACM.

[4] Amy Henderson Law, Jim Ahrens, and Charles, *The ParaView Guide*, Kitware Inc., Clifton Park, NY., 2004.

[5] Hank Childs Max, Eric S. Brugger, Kathleen S. Bonnell, Jeremy S. Meredith, Mark Miller, Brad J. Whitlock, and Nelson, "A Contract-Based System for Large Data Visualization," in *Proceedings of IEEE Visualization 2005*, 2005, pp. 190–198.

[6] William E. Lorensen and Harvey E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 1987, SIGGRAPH '87, pp. 163–169, ACM.

[7] Florian Lindemann and Timo Ropinski, "About the Influence of Illumination Models on Image Comprehension in Direct Volume Rendering," *IEEE TVCG(Vis Proceedings)*, vol. 17, no. 12, pp. 1922–1931, 2011.

[8] Mathias Schott, Vincent Pegoraro, Charles D. Hansen, Kevin Boulanger, and Kadi Bouatouch, "A Directional Occlusion Shading Model for Interactive Direct Volume Rendering.," *Comput. Graph. Forum*, vol. 28, no. 3, pp. 855–862, 2009.

[9] Christian Boucheny, Georges-Pierre Bonneau, Jacques Droulez, Guillaume Thibault, and Stephane Ploix, "A perceptive evaluation of volume rendering techniques," *ACM Trans. Appl. Percept.*, vol. 5, no. 4, pp. 23:1–23:24, Feb. 2009.

[10] Timo Ropinski, Frank Steinicke, and Klaus Hinrichs, "Visually supporting depth perception in angiography imaging," in *Smart Graphics*. 2006, pp. 93–104, Springer.

[11] William M. Hsu, "Segmented Ray Casting for Data Parallel Volume Rendering," in *Proceedings of the 1993 Symposium on Parallel Rendering*, New York, NY, USA, 1993, PRS '93, pp. 7–14, ACM.

[12] K.-L. Ma, J.S. Painter, C.D. Hansen, and M.F. Krogh, "A data distributed, parallel algorithm for ray-traced volume rendering," in *Parallel Rendering Symposium, 1993*, 1993, pp. 15–22, 105.

[13] Tom Peterka, David Goodell, Robert Ross, Han-Wei Shen, and Rajeev Thakur, "A Configurable Algorithm for Parallel Image-compositing Applications," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, 2009, SC '09, pp. 4:1–4:10, ACM.

[14] M. Howison, E.W. Bethel, and H. Childs, "Hybrid Parallelism for Volume Rendering on Large-, Multi-, and Many-Core Systems," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, no. 1, pp. 17–29, Jan 2012.

[15] Kenneth Moreland, "IceT Users' Guide and Reference," Tech. Rep., Sandia National Lab, January 2011.

[16] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey, "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 451–460, June 2010.

[17] CIBC, ," 2015, ImageVis3D: An interactive visualization software system for large-scale volume data. Scientific Computing and Imaging Institute (SCI), Download from: http://www.imagevis3d.org.

[18] Thomas Fogal and Jens Kruger, "Tuvok, an Architecture for Large Scale Volume Rendering," in *Proceedings of the 15th International Workshop on Vision, Modeling, and Visualization*, November 2010.

[19] A.V.P. Grosset, M. Prasad, C Christensen, A Knoll, and C.D. Hansen, "TOD-Tree: Task-Overlapped Direct send Tree Image Compositing for Hybrid MPI Parallelism," in *Eurographics Symposium on Parallel Graphics and Visualization*, 2015.

[20] A. V. P. Grosset, M. Schott, G. P. Bonneau, and C. D. Hansen, "Evaluation of depth of field for depth perception in dvr," in *Visualization Symposium (PacificVis), 2013 IEEE Pacific*, Feb 2013, pp. 81–88.

[21] A. V. P. Grosset, M. Prasad, C. Christensen, A. Knoll, and C. Hansen, "Tod-tree: Task-overlapped direct send tree image compositing for hybrid mpi parallelism and gpus," *IEEE Transactions on Visualization and Computer Graphics*, vol. PP, no. 99, pp. 1–1, 2016.

[22] A.V.P. Grosset, A Knoll, and C.D. Hansen, "Dynamically Scheduled Region-Based Image Compositing," in *Eurographics Symposium on Parallel Graphics and Visualization*, 2016.

[23] Rogers Brian J. Howard Ian P., *Seeing in Depth*, vol. 2 Depth Perception, I Porteous, 2002.

[24] William Thompson, Roland Fleming, Sarah Creem-Regehr, and Jeanine Kelly Stefanucci, *Visual Perception from a Computer Graphics Perspective*, A. K. Peters, Ltd., Natick, MA, USA, 1st edition, 2011.

[25] George Mather and David R.R Smith, "Blur discrimination and its relation to blur-mediated depth perception," *Perception*, vol. 31(10), pp. 1211–1219, 2002.

[26] Brian A. Barsky and Todd J. Kosloff, "Algorithms for rendering depth of field effects in computer graphics," in *Proceedings of the 12th WSEAS international conference on Computers*, Stevens Point, Wisconsin, USA, 2008, ICCOMP'08, pp. 999–1010, World Scientific and Engineering Academy and Society (WSEAS).

[27] Michael Potmesil and Indranil Chakravarty, "A lens and aperture camera model for synthetic image generation," in *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 1981, SIGGRAPH '81, pp. 297–305, ACM.

[28] Robert L. Cook, Thomas Porter, and Loren Carpenter, "Distributed ray tracing," *SIGGRAPH Comput. Graph.*, vol. 18, no. 3, pp. 137–145, Jan. 1984.

[29] Brian A. Barsky, Daniel R. Horn, Stanley A. Klein, Jeffrey A. Pang, and Meng Yu, "Camera Models and Optical Systems Used in Computer Graphics: Part II, Image-Based Techniques," in *Computational Science and Its Applications - ICCSA 2003*, Lecture Notes in Computer Science, pp. 256–265. Springer Berlin / Heidelberg, 2003.

[30] Martin Kraus and Magnus Strengert, "Depth-of-Field Rendering by Pyramidal Image Processing," *Computer Graphics Forum*, vol. 26, pp. 645–654, Sep 2007.

[31] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann, "GigaVoxels: ray-guided streaming for efficient and detailed voxel rendering," in *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. 2009, I3D '09, pp. 15–22, ACM.

[32] Robert T. Held, Emily A. Cooper, and Martin S. Banks, "Blur and Disparity Are Complementary Cues to Depth," *Current Biology*, vol. 22, no. 5, pp. 426–431, 2012.

[33] George Mather and David R.R Smith, "Combining depth cues: effects upon accuracy and speed of performance in a depth-ordering task," *Vision Research*, vol. 44, no. 6, pp. 557–562, 2004.

[34] Robert T. Held, Emily A. Cooper, James F. O'Brien, and Martin S. Banks, "Using blur to affect perceived distance and size," *ACM Trans. Graph.*, vol. 29, no. 2, pp. 19:1–19:16, Apr. 2010.

[35] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, "A sorting classification of parallel rendering," *Computer Graphics and Applications, IEEE*, vol. 14, no. 4, pp. 23–32, 1994.

[36] Ulrich Neumann, "Communication Costs for Parallel Volume-Rendering Algorithms," *IEEE Comput. Graph. Appl.*, vol. 14, no. 4, pp. 49–58, July 1994.

[37] Stefan Eilemann and Renato Pajarola, "Direct Send Compositing for Parallel Sort-last Rendering," in *Proceedings of the 7th Eurographics Conference on Parallel Graphics and Visualization*, Aire-la-Ville, Switzerland, Switzerland, 2007, EG PGV'07, pp. 29–36, Eurographics Association.

[38] Silvio Rizzi, Mark Hereld, Joseph Insley, Michael E. Papka, Thomas Uram, and Venkatram Vishwanath, "Performance Modeling of vl3 Volume Rendering on GPU-Based Clusters," in *Eurographics Symposium on Parallel Graphics and Visualization*, Margarita Amor and Markus Hadwiger, Eds. 2014, The Eurographics Association.

[39] Christopher D. Shaw, Mark Green, and Jonathan Schaeffer, "Advances in Computer Graphics Hardware III," chapter A VLSI Architecture for Image Composition, pp. 183–199. Springer-Verlag New York, Inc., New York, NY, USA, 1991.

[40] Hongfeng Yu, Chaoli Wang, and Kwan-Liu Ma, "Massively Parallel Volume Rendering Using 2-3 Swap Image Compositing," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Piscataway, NJ, USA, 2008, SC '08, pp. 48:1–48:11, IEEE Press.

[41] Kenneth Moreland, Wesley Kendall, Tom Peterka, and Jian Huang, "An Image Compositing Solution at Scale," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2011, SC '11, pp. 25:1–25:10, ACM.

[42] M. Howison, E. W. Bethel, and H. Childs, "Mpi-hybrid parallelism for volume rendering on large, multi-core systems," in *Proceedings of the 10th Eurographics Conference on Parallel Graphics and Visualization*, Aire-la-Ville,

Switzerland, Switzerland, 2010, EG PGV'10, pp. 1–10, Eurographics Association.

[43] Aleksander Stompel, Kwan-Liu Ma, Eric B. Lum, James Ahrens, and John Patchett, "Slic: Scheduled linear image compositing for parallel volume rendering," in *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, Washington, DC, USA, 2003, PVG '03, pp. 6–, IEEE Computer Society.

[44] Magnus Strengert, Marcelo Magalln, Daniel Weiskopf, Stefan Guthe, and Thomas Ertl, "Hierarchical Visualization and Compression of Large Volume Datasets Using GPU Clusters," in *Eurographics Workshop on Parallel Graphics and Visualization*, Dirk Bartz, Bruno Raffin, and Han-Wei Shen, Eds. 2004, The Eurographics Association.

[45] Brendan Moloney, Daniel Weiskopf, Torsten Möller, and Magnus Strengert, "Scalable sort-first parallel direct volume rendering with dynamic load balancing," in *Proceedings of the 7th Eurographics Conference on Parallel Graphics and Visualization*, Aire-la-Ville, Switzerland, Switzerland, 2007, EGPGV '07, pp. 45–52, Eurographics Association.

[46] C. Müller, M. Strengert, and T. Ertl, "Adaptive load balancing for raycasting of non-uniformly bricked volumes," *Parallel Comput.*, vol. 33, no. 6, pp. 406–419, June 2007.

[47] D. Pugmire, L. Monroe, C. Connor Davenport, A. DuBois, D. DuBois, and S. Poole, "Npu-based image compositing in a distributed visualization system," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 4, pp. 798–809, July 2007.

[48] Xavier Cavin and Olivier Demengeon, "Shift-Based Parallel Image Compositing on InfiniBand TM Fat-Trees," in *Eurographics Symposium on Parallel Graphics and Visualization*, Hank Childs, Torsten Kuhlen, and Fabio Marton, Eds. 2012, The Eurographics Association.

[49] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski, "Chromium: A Stream-processing Framework for Interactive Rendering on Clusters," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 693–702, July 2002.

[50] Stefan Eilemann, Maxim Makhinya, and Renato Pajarola, "Equalizer: A Scalable Parallel Rendering Framework," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 3, pp. 436–452, May 2009.

[51] Timothy J. Cullip and Ulrich Neumann, "Accelerating Volume Reconstruction With 3D Texture Hardware," Tech. Rep., Chapel Hill, NC, USA, 1994.

[52] J. Kruger and R. Westermann, "Acceleration Techniques for GPU-based Volume Rendering," in *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, Washington, DC, USA, 2003, VIS '03, pp. 38–, IEEE Computer Society.

[53] C. Müller, M. Strengert, and T. Ertl, "Optimized Volume Raycasting for Graphics-hardware-based Cluster Systems," in *Proceedings of the 6th Eurographics Conference on Parallel Graphics and Visualization*, Aire-la-Ville, Switzerland, Switzerland, 2006, EG PGV'06, pp. 59–67, Eurographics Association.

[54] Thomas Fogal, Hank Childs, Siddharth Shankar, Jens Krüger, R. Daniel Bergeron, and Philip Hatcher, "Large Data Visualization on Distributed Memory multi-GPU Clusters," in *Proceedings of the Conference on High Performance Graphics*, Aire-la-Ville, Switzerland, Switzerland, 2010, HPG '10, pp. 57–66, Eurographics Association.

[55] Jinrong Xie, Hongfeng Yu, and Kwan-Liu Ma, "Visualizing large 3D geodesic grid data with massively distributed GPUs," in *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, Nov 2014, pp. 3–10.

[56] Hao Wang, Sreeram Potluri, Miao Luo, Ashish Kumar Singh, Sayantan Sur, and Dhabaleswar K. Panda, "MVAPICH2-GPU: Optimized GPU to GPU Communication for InfiniBand Clusters," *Comput. Sci.*, vol. 26, no. 3-4, pp. 257–266, June 2011.

[57] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D.K. Panda, "Efficient Inter-node MPI Communication Using GPUDirect RDMA for InfiniBand Clusters with NVIDIA GPUs," in *Parallel Processing (ICPP), 2013 42nd International Conference on*, Oct 2013, pp. 80–89.

[58] Alex James, "An introduction to gpudirect," November 2015.

[59] Nvidia, "Remote Visualization on Server-Class Tesla GPUs," White Paper WP-07313-001-v01, Nvidia, June 2014.

[60] Mark D. Klein and John E. Stone, "Unlocking the Full Potential of the Cray XK7 Accelerator Mark," in *Cray User Group Conference*. Cray, May 2014.

[61] Daniel Jönsson, Erik Sundén, Anders Ynnerman, and Timo Ropinski, "A survey of volumetric illumination techniques for interactive volume rendering," *Comput. Graph. Forum*, vol. 33, no. 1, pp. 27–51, Feb. 2014.

[62] D.C. Knill, "Reaching for visual cues to depth: The brain combines depth cues differently for motor control and perception," *Journal of Vision*, vol. 5, no. 2, 2005.

[63] Jonathan W. Peirce, "PsychoPy—Psychophysics software in Python," *Journal of Neuroscience Methods*, vol. 162, no. 1–2, pp. 8–13, 2007.

[64] A. A. Michelson, *Studies in Optics*, Univ. Chicago Press, Chicago, IL, 1927.

[65] Ben Caldwell, Michael Cooper, Loretta Guarino Reid, and Gregg Vanderheiden, "Web Content Accessibility Guidelines (WCAG) 2.0," `http://www.w3.org/TR/WCAG20/`, 12 2008.

[66] R. Kosara, C.G. Healey, V. Interrante, D.H. Laidlaw, and C. Ware, "Visualization viewpoints," *Computer Graphics and Applications, IEEE*, vol. 23, no. 4, pp. 20–25, july-aug. 2003.

[67] Luca Filippin, "T2tpkg," `https://sites.google.com/site/t2tpkg/`.

[68] Tobii Technology, "Tobii T60 and T120 Eye Tracker," `http://www.tobii.com/Global/Analysis/Downloads/User\_Manuals\_and\_Guides/Tobii\_T60\_T120\_EyeTracker\_UserManual.pdf`.

[69] Rafal Bogacz, Eric Brown, Jeffrey Moehlis, Phil Holmes, and Jonathan D. Cohen, "The physics of optimal decision making: A formal analysis of models of performance in two-alternative forced choice tasks," *Psychological Review*, vol. 113, no. 4, pp. 700–765, October 2006.

[70] David Drascic and Paul Milgram, "Perceptual issues in augmented reality," in *SPIE Volume 2653: Stereoscopic Displays and Virtual Reality Systems III*, 1996, pp. 123–134.

[71] Agnieszka Wykowska, Anna Schubo, and Bernhard Hommel, "How You Move Is What You See: Action Planning Biases Selection in Visual Search," *Journal of Experimental Psychology: Human Perception and Performance*, vol. 35, no. 6, pp. 1755–1769, 2009.

[72] Steve Ashby, Pete Beckman, Jackie Chen, Phil Colella, Bill Collins, Dona Crawford, Jack Dongarra, Doug Kothe, Rusty Lusk, Paul Messina, and Others, "The opportunities and challenges of exascale computing.," *summary report of the advanced scientific computing advisory committee (ASCAC) subcommittee at the US Department of Energy Office of Science*, 2010.

[73] Damián A. Mallón, Guillermo L. Taboada, Carlos Teijeiro, Juan Touriño, Basilio B. Fraguela, Andrés Gómez, Ramón Doallo, and J. Carlos Mouriño, "Performance Evaluation of MPI, UPC and OpenMP on Multicore Architectures," in *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Berlin, Heidelberg, 2009, pp. 174–184, Springer-Verlag.

[74] Rolf Rabenseifner, Georg Hager, and Gabriele Jost, "Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes," in *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Washington, DC, USA, 2009, PDP '09, pp. 427–436, IEEE Computer Society.

[75] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn, "Collective Communication: Theory, Practice, and Experience: Research Articles," *Concurr. Comput. : Pract. Exper.*, vol. 19, no. 13, pp. 1749–1783, Sept. 2007.

[76] NERSC, "Edison Configuration," February 2015.

[77] TACC, "Stampede User Guide," February 2015.

[78] Leonardo Dagum and Ramesh Menon, "OpenMP: An Industry-Standard API for Shared-Memory Programming," *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Jan. 1998.

[79] Stéphane Marchesin, Catherine Mongenet, and Jean-Michel Dischler, "Multi-GPU Sort-last Volume Visualization," in *Proceedings of the 8th Eurographics Conference on Parallel Graphics and Visualization*, Aire-la-Ville, Switzerland, Switzerland, 2008, EGPGV '08, pp. 1–8, Eurographics Association.

[80] Peter Messmer, "Egl eye: Opengl visualization without an x server," January 2016.

[81] NVIDIA, "NVIDIA TESLA GPU ACCELERATORS," October 2015.

[82] CSCS, "Piz Daint," October 2015.

[83] Kenneth Moreland, Brian N. Wylie, and Constantine J. Pavlakos, "Sort-last parallel rendering for viewing extremely large data sets on tile displays.," in *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, David E. Breen, Alan Heirich, and Anton H. J. Koning, Eds. 2001, pp. 85–92, IEEE.

[84] Michael Potmesil and Indranil Chakravarty, "A lens and aperture camera model for synthetic image generation," in *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 1981, SIGGRAPH '81, pp. 297–305, ACM.

[85] "GPU Direct RDMA," .

[86] Felix Manke and Burkhard Wünsche, "Texture-enhanced Direct Volume Rendering," in *GRAPP*, 2009, pp. 185–190.

[87] Daniel S. Schlusselberg, Wade K. Smith, and Doanld J. Woodward, "Three-Dimensional Display of Medical Image Volumes," *Proceedings of NCGA*, March 1986.

[88] Peter Rautek, Stefan Bruckner, Eduard Gröller, and Ivan Viola, "Illustrative Visualization: New Technology or Useless Tautology?," *SIGGRAPH Comput. Graph.*, vol. 42, no. 3, pp. 4:1–4:8, Aug. 2008.

[89] Edward H. Adelson and P. An, "Ordinal characteristics of transparency," in *in Proc. AAAI workshop on Qualitative Vision*, 1990, pp. 77–81.

[90] P Alliez and K Bala, "Real-time Depth of Field Rendering Via Dynamic Light Field Generation and Filtering," *eecis.udel.edu*, vol. 29, no. 7, 2010.

[91] Brian Austin Wright, Matthew J. Cordery, Harvey J. Wasserman, and Nicholas J., *Performance measurements of the nersc cray cascade system*, Cray, Inc., May 2013.

[92] Brian A. Barsky, Daniel R. Horn, Stanley A. Klein, Jeffrey A. Pang, and Meng Yu, "Camera models and optical systems used in computer graphics: Part I, Object based techniques," in *Computational Science and Its Applications - ICCSA 2003*, Lecture Notes in Computer Science, pp. 246–255. Springer Berlin / Heidelberg, 2003.

[93] Jon Louis Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sept. 1975.

[94] M. J. Berger and P. Colella, "Local Adaptive Mesh Refinement for Shock Hydrodynamics," *J. Comput. Phys.*, vol. 82, no. 1, pp. 64–84, May 1989.

[95] Marsha J. Berger and Joseph E. Oliger, "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations," Tech. Rep., Stanford, CA, USA, 1983.

[96] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, Peter Kogge, R. Stanley Williams, and Katherine Yelick, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," Tech. Rep., 2008.

[97] Martin Berzins, Justin Luitjens, Qingyu Meng, Todd Harman, Charles A. Wight, and Joseph R. Peterson, "Uintah: A Scalable Framework for Hazard Analysis," in *Proceedings of the 2010 TeraGrid Conference*, New York, NY, USA, 2010, TG '10, pp. 3:1–3:8, ACM.

[98] E. Wes Bethel, Hank Childs, and Charles Hansen, *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*, Chapman & Hall/CRC, 1st edition, 2012.

[99] Stefan Bruckner, Sören Grimm, Armin Kanitsar, and M Eduard Gröller, "Illustrative Context-Preserving Volume Rendering," *Direct*, vol. 1, 2005.

[100] X. Cavin, C. Mion, and A. Filbois, "COTS cluster-based sort-last rendering: performance evaluation and pipelined implementation," in *Visualization, 2005. VIS 05. IEEE*, Oct 2005, pp. 111–118.

[101] Hank Childs, Mark Duchaineau, and Kwan-Liu Ma, "A Scalable, Hybrid Scheme for Volume Rendering Massive Data Sets," in *Proceedings of the 6th Eurographics Conference on Parallel Graphics and Visualization*, Aire-la-Ville, Switzerland, Switzerland, 2006, EG PGV'06, pp. 153–161, Eurographics Association.

[102] Kenneth J. Ciuffreda, Bin Wang, and Balamurali Vasudevan, "Conceptual model of human blur perception," *Vision Research*, vol. 47, no. 9, pp. 1245–1252, 2007.

[103] M. Bertalmio, P. Fort, and D. Sanchez-Crespo, "Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards," in *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004.* 2004, pp. 767–773, Ieee.

[104] M. Cohen and K. Brodlie, "Focus and context for volume visualization," *Proceedings Theory and Practice of Computer Graphics, 2004.*, vol. 6, pp. 32–39, 2004.

[105] Carlos Correa, Debora Silver, and Min Chen, "Illustrative deformation for data exploration.," *IEEE transactions on visualization and computer graphics*, vol. 13, no. 6, pp. 1320–7, 2007.

[106] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, Miguel Sainz, and Elmar Eisemann, "Beyond triangles: gigavoxels effects in video games," in *SIGGRAPH 2009: Talks*. 2009, SIGGRAPH '09, pp. 78:1–78:1, ACM.

[107] Joe Demers, *GPU Gems*, chapter 23. Depth of Field: A Survey of Techniques, pp. 375–390, Addison-Wesley Longman, Inc., 2004.

[108] Louis Derr, *Photography for students of physics and chemistry*, chapter 6, p. 79, Macmillan & Co., Ltd, 1906.

[109] Kai-Uwe Doerr and Falko Kuester, "CGLX: A Scalable, High-Performance Visualization Framework for Networked Display Environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 3, pp. 320–332, May 2011.

[110] David Ebert and Penny Rheingans, "Volume illustration: non-photorealistic rendering of volume models," in *Proceedings of the conference on Visualization '00*, Los Alamitos, CA, USA, 2000, VIS '00, pp. 195–202, IEEE Computer Society Press.

[111] Markus Hadwiger, Joe M. Kniss, Christof Rezk-salama, Daniel Weiskopf, and Klaus Engel, *Real-time Volume Graphics*, A. K. Peters, Ltd., Natick, MA, USA, 2006.

[112] Wei Fang, Guangzhong Sun, Peng Zheng, Tiening He, and Guoliang Chen, "Efficient Pipelining Parallel Methods for Image Compositing in Sort-last Rendering," in *Proceedings of the 2010 IFIP International Conference on Network and Parallel Computing*, Berlin, Heidelberg, 2010, NPC'10, pp. 289–298, Springer-Verlag.

[113] Randima Fernando, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, Pearson Higher Education, 2004.

[114] Thomas Fogal and Jens Krüger, "Efficient I/O for Parallel Visualization," in *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, Aire-la-Ville, Switzerland, Switzerland, 2011, EG PGV'11, pp. 81–90, Eurographics Association.

[115] Thomas Fogal, Alexander Schiewe, and Jens Kruger, "An analysis of scalable GPU-based ray-guided volume rendering," in *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*, 2013, pp. 43–51.

[116] Sören Grimm, Stefan Bruckner, Armin Kanitsar, and Eduard Gröller, "A refined data addressing and processing scheme to accelerate volume raycasting," *Computers Graphics*, vol. 28, no. 5, pp. 719–729, 2004.

[117] Markus Hadwiger, Joe M. Kniss, Christof Rezk-salama, Daniel Weiskopf, and Klaus Engel, *Real-time Volume Graphics*, A. K. Peters, Ltd., Natick, MA, USA, 2006.

[118] Paul Haeberli and Kurt Akeley, "The accumulation buffer: hardware support for high-quality rendering," *SIGGRAPH Comput. Graph.*, vol. 24, pp. 309–318, September 1990.

[119] H. Hauser, L. Mroz, G. Italo Bischi, and M.E. Groller, "Two-level volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, no. 3, pp. 242–252, 2001.

[120] Milan Ikits and C.D. Hansen, "A focus and context interface for interactive volume rendering," *Unpublished Work*, 2004.

[121] Victoria Interrante, Henry Fuchs, and Stephen M. Pizer, "Illustrating Transparent Surfaces with Curvature-Directed Strokes," in *Proceedings of IEEE Visualization 1996*, 1996, pp. 211–218, 487.

[122] Demers Joe, *GPU Gems*, vol. 1, chapter 23 Depth of Field: A Survey of Techniques, Addison-Wesley Professional, 2004.

[123] Ralf Kähler and Tom Abel, "Single-Pass GPU-Raycasting for Structured Adaptive Mesh Refinement Data," *CoRR*, vol. abs/1212.3333, 2012.

[124] R. Kaehler, S. Prohaska, A. Hutanu, and H.-C. Hege, "Visualization of time-dependent remote adaptive mesh refinement data," in *Visualization, 2005. VIS 05. IEEE*, 2005, pp. 175–182.

[125] M Kass and A Lefohn, "Interactive depth of field using simulated diffusion on a gpu," *Pixar Animation Studios, Tech. Rep*, 2006.

[126] Darren J. Kerbyson, Kevin J. Barker, Abhinav Vishnu, and Adolfy Hoisie, "A Performance Comparison of Current HPC Systems: Blue Gene/Q, Cray XE6 and InfiniBand Systems," *Future Gener. Comput. Syst.*, vol. 30, pp. 291–304, Jan. 2014.

[127] Darren J. Kerbyson, Kevin J. Barker, Abhinav Vishnu, and Adolfy Hoisie, "Comparing the Performance of Blue Gene/Q with Leading Cray XE6 and InfiniBand Systems," in *Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems*, Washington, DC, USA, 2012, ICPADS '12, pp. 556–563, IEEE Computer Society.

[128] Robert Kosara, *Semantic Depth of Field - Using Blur for Focus+Context Visualization*, Ph.D. thesis, Vienna University of Technology, Vienna, Austria, 2001.

[129] Robert Kosara, Silvia Miksch, and Helwig Hauser, "Semantic Depth of Field," in *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, Washington, DC, USA, 2001, INFOVIS '01, pp. 97–, IEEE Computer Society.

[130] Robert Kosara, Silvia Miksch, H. Hauser, Johann Schrammel, Verena Giller, and Manfred Tscheligi, "Useful properties of semantic depth of field for better f+ c visualization," in *Proceedings of the symposium on Data Visualisation 2002*. 2002, pp. 205–210, Eurographics Association.

[131] Robert Kosara, Silvia Miksch, Helwig Hauser, Johann Schrammel, Verena Giller, and Manfred Tscheligi, "Useful properties of Semantic Depth of Field for better F+C visualization," in *Proceedings of the symposium on Data Visualisation 2002*, Aire-la-Ville, Switzerland, Switzerland, 2002, VISSYM '02, pp. 205–210, Eurographics Association.

[132] T.J. Kosloff and B.A. Barsky, "Three Techniques for Rendering Generalized Depth of Field Effects," *siam.org*, pp. 42–48, 2009.

[133] J. Kr\\"uger and T Fogal, "Focus and Context-Visualization without the Complexity," in *World Congress on Medical Physics and Biomedical Engineering, September 7-12, 2009, Munich, Germany*. 2009, pp. 45–48, Springer.

[134] Oliver Kreylos, Gunther H. Weber, E. Wes Bethel, E. Wes, Bethel John, John M. Shalf, Bernd Hamann, and Kenneth I. Joy, "Remote Interactive Direct Volume Rendering of AMR Data," 2002.

[135] TOP 500, "Top 500 List - June 2015," October 2015.

[136] J. Krivanek, J. Zara, and K. Bouatouch, "Fast depth of field rendering with surface splatting," *Proceedings Computer Graphics International 2003*, pp. 196–201, 2003.

[137] Eric C. La Mar, Bernd Hamann, and Kenneth I. Joy, "Multiresolution Techniques for Interactive Texture-based Volume Visualization," in *Proceedings of the 10th IEEE Visualization 1999 Conference (VIS '99)*, Washington, DC, USA, 1999, VISUALIZATION '99, pp. –, IEEE Computer Society.

[138] Nick Leaf, "Efficient Parallel Volume Rendering of Large-Scale Adaptive Mesh Refinement Data," .

[139] Marc Levoy, "Efficient Ray Tracing of Volume Data," *ACM Trans. Graph.*, vol. 9, no. 3, pp. 245–261, July 1990.

[140] Bo Li and Hong Qin, "Feature-Aware Reconstruction of Volume Data via Trivariate Splines," 2011, pp. 49–54, Pacific Graphics 2011.

[141] F. Lindemann and T. Ropinski, "About the Influence of Illumination Models on Image Comprehension in Direct Volume Rendering," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 12, pp. 1922–1931, dec. 2011.

[142] Shusen Liu, V Vishwanath, J. A Insley, M Hereld, M. E Papka, and Valerio Pascucci, "A Static Load Balancing Scheme for Parallel Volume Rendering on Multi-GPU Clusters," in *Large-Scale Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*. 2012, HGPU Group.

[143] LLNL, "VisIt Visualization Tool," Aug. 2012.

[144] Eric Lum Patchett, Kwan-Liu Ma, James Ahrens, and John, "SLIC: Scheduled Linear Image Compositing for Parallel Vollume Rendering," Parallel Visualization and Graphics 2003, 2003, IEEE.

[145] Kwan-Liu Ma, "Parallel Rendering of 3D AMR Data on the SGI/Cray T3E," in *Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation*, Washington, DC, USA, 1999, FRONTIERS '99, pp. 138–, IEEE Computer Society.

[146] S. Marchesin and G.C. de Verdiere, "High-Quality, Semi-Analytical Volume Rendering for AMR Data," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 6, pp. 1611–1618, 2009.

[147] Jonathan A. Marshall, Christina A. Burbeck, Dan Ariely, Jannick P. Rolland, and Kevin E. Martin, "Occlusion edge blur: a cue to relative visual depth," *J. Opt. Soc. Am. A*, vol. 13, no. 4, pp. 681–688, Apr 1996.

[148] Jonathan A. Marshall, Jozathaz A. Marshall, Dan Ariely, Christina A. Burbeck, T Daz Aricly, Jannick P. Rolland, and Kevin E. Martin, "Occlusion edge blur: A cue to relative visual depth," *Intl. J. Opt. Soc. Am. A*, vol. 13, pp. 681–688, 1996.

[149] K.E. Martin, D.H. Dawes, and R.E. Faith, "Distributed Multihead X Design," July 2001.

[150] R. Martinez-Cuenca, G. Saavedra, M. Martinez-Corral, and B. Javidi, "Extended Depth-of-Field 3-D Display and Visualization by Combination of Amplitude-Modulated Microlenses and Deconvolution Tools," *Journal of Display Technology*, vol. 1, no. 2, pp. 321–327, Dec. 2005.

[151] George Mather, "Image Blur as a Pictorial Depth Cue," *Proceedings: Biological Sciences*, vol. 263, no. 1367, pp. pp. 169–172, 1996.

[152] Manabu Matsui, Fumihiko Ino, and Kenichi Hagihara, "Parallel Volume Rendering with Early Ray Termination for Visualizing Large-Scale Datasets," in *Parallel and Distributed Processing and Applications*, Jiannong Cao, LaurenceT. Yang, Minyi Guo, and Francis Lau, Eds., vol. 3358 of *Lecture Notes in Computer Science*, pp. 245–256. Springer Berlin Heidelberg, 2005.

[153] Manabu Matsui, Fumihiko Ino, and Kenichi Hagihara, "Parallel Volume Rendering with Early Ray Termination for Visualizing Large-scale Datasets," in *Proceedings of the Second International Conference on Parallel and Distributed Processing and Applications*, Berlin, Heidelberg, 2004, ISPA'04, pp. 245–256, Springer-Verlag.

[154] K. J. McCauley, S. A. Moorman, and D. K. McDonald, "Oxy-Coal Combustion for Low Carbon Electric Power Generation," Tech. Rep., May 2011.

[155] J. Meyer-Spradow, Timo Ropinski, and Klaus Hinrichs, "Supporting Depth and Motion Perception in Medical Volume Data," *Visualization in Medicine and Life Sciences*, vol. D, pp. 121–133, 2008.

[156] B. Moloney, M. Ament, D. Weiskopf, and T. Moller, "Sort-First Parallel Volume Rendering," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 8, pp. 1164–1177, Aug 2011.

[157] Mark Mon-Williams and James R. Tresilian, "Ordinal depth information from accommodation?," *Ergonomics*, vol. 43, no. 3, pp. 391–404, 2000.

[158] P.J. Moran and D. Ellsworth, "Visualization of AMR Data With Multi-Level Dual-Mesh Interpolation," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 12, pp. 1862–1871, 2011.

[159] Jurriaan D. Mulder and Robert van Liere, "Fast perception-based depth of field rendering," *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '00*, p. 129, 2000.

[160] Vincent A. Nguyen, Ian P. Howard, and Robert S. Allison, "Detection of the depth order of defocused images," *Vision Research*, vol. 45, no. 8, pp. 1003–1011, 2005.

[161] M. L. Norman, J. M. Shalf, S. Levy, and G. Daues, "Diving deep: Data management and visualization strategies for adaptive mesh refinement simulations," *Computing in Science and Engineering*, vol. 1, no. 4, pp. 36–47, 1999.

[162] Kevin L. Novins, Francois X. Sillion, and Donald P. Greenberg, "An efficient method for volume rendering using perspective projection," *SIGGRAPH Comput. Graph.*, vol. 24, pp. 95–102, November 1990.

[163] Sanghun Park, Chandrajit L. Bajaj, and Vinay Siddavanahalli, "Case Study: Interactive Rendering of Adaptive Mesh Refinement Data," in *Proceedings of the Conference on Visualization '02*, Washington, DC, USA, 2002, VIS '02, pp. 521–524, IEEE Computer Society.

[164] Michael Potmesil and Indranil Chakravarty, "Synthetic Image Generation with a Lens and Aperture Camera Model," *ACM Trans. Graph.*, vol. 1, no. 2, pp. 85–108, Apr. 1982.

[165] Michael Potmesil and Indranil Chakravarty, "A lens and aperture camera model for synthetic image generation," in *Proceedings of the 8th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1981, SIGGRAPH '81, pp. 297–305, ACM.

[166] D.I.P. Rautek, "Semantic Visualization Mapping for Volume Illustration," vol. Ill, 2008.

[167] Sidney F. Ray, *Applied Photographic Optics: Lenses and Optical Systems for Photography, Film, Video and Electronic Imaging*, Focal Press, 1994.

[168] Austin Robison and Peter Shirley, "Image space gathering," *Proceedings of the 1st ACM conference on High Performance Graphics - HPG '09*, p. 91, 2009.

[169] Przemyslaw Rokita, "Generating Depth-of-Field Effects in Virtual Reality Applications," *IEEE Computer Graphics and Applications*, vol. 16, pp. 18–21, 1996.

[170] Carlos Rosales, "ACELab Report: Stampede Baseline Evaluation," TACC Technical Report TR-14-08, TACC, Advanced Computing Evaluation Laboratory, Texas Advanced Computing Center, The University of Texas at Austin, September 2014.

[171] Cary Scofield, *2 1/2-D depth-of-field simulation for computer animation*, pp. 36–38, Academic Press Professional, Inc., San Diego, CA, USA, 1992.

[172] John Shalf, Sudip Dosanjh, and John Morrison, "Exascale Computing Technology Challenges," in *Proceedings of the 9th International Conference on High Performance Computing for Computational Science*, Berlin, Heidelberg, 2011, VECPAR'10, pp. 1–25, Springer-Verlag.

[173] Jeff A. Stuart, Cheng-Kai Chen, Kwan-Liu Ma, and John D. Owens, "Multi-GPU Volume Rendering Using MapReduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, New York, NY, USA, 2010, HPDC '10, pp. 841–848, ACM.

[174] Akira Takeuchi, Fumihiko Ino, and Kenichi Hagihara, "An Improved Binary-swap Compositing for Sort-last Parallel Rendering on Distributed Memory Multiprocessors," *Parallel Comput.*, vol. 29, no. 11-12, pp. 1745–1762, Nov. 2003.

[175] 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*, vol. 192, no. 1, January 2011.

[176] I. Viola, A. Kanitsar, and M.E. Groller, "Importance-driven volume rendering," in *Visualization, 2004. IEEE*, Oct. 2004, pp. 139–145.

[177] Huy T. Vo, Steven P. Callahan, Nathan Smith, Cláudio T. Silva, William Martin, David Owen, and David Weinstein, "Interactive Rendering of Large Unstructured Grids," in *Proceedings of the 7th Eurographics Conference on Parallel Graphics and Visualization*, Aire-la-Ville, Switzerland, Switzerland, 2007, EG PGV'07, pp. 93–100, Eurographics Association.

[178] J W Wallis and T R Miller, "Volume rendering in three-dimensional display of SPECT images.," *Journal of nuclear medicine : official publication, Society of Nuclear Medicine*, vol. 31, no. 8, pp. 1421–8, Aug. 1990.

[179] L. Wang, Y. Zhao, K. Mueller, and A. Kaufman, "The magic volume lens: an interactive focus+context technique for volume rendering," in *Visualization, 2005. VIS 05. IEEE*, Oct. 2005, pp. 367–374.

[180] Yu-Shuen Wang, Chaoli Wang, Tong-Yee Lee, and Kwan-Liu Ma, "Feature-Preserving Volume Data Reduction and Focus+Context Visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 2, pp. 171–181, Feb. 2011.

[181] Simon J. Watt, Kurt Akeley, Marc O. Ernst, and Martin S. Banks, "Focus cues affect perceived depth," *Journal of Vision*, vol. 5, no. 10, 2005.

[182] Gunther H. Weber, Martin Öhler, Oliver Kreylos, John Shalf, E. Wes Bethel, Bernd Hamann, and Gerik Scheuermann, "Parallel Cell Projection Rendering of Adaptive Mesh Refinement Data.," in *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, Anton H. J. Koning, Raghu Machiraju, and Cláudio T. Silva, Eds. 2003, pp. 51–60, IEEE.

[183] Gunther H. Weber, Hank Childs, and Jeremy S. Meredith, "Efficient parallel extraction of crack-free isosurfaces from adaptive mesh refinement (AMR) data.," in *LDAV*, Roger S. Barga, Hanspeter Pfister, and David Rogers, Eds. 2012, pp. 31–38, IEEE.

[184] J.J. van Wijk, "Flow visualization with surface particles," *IEEE Computer Graphics and Applications*, vol. 13, no. 4, pp. 18–24, July 1993.

[185] Don-Lin Yang, Jen-Chih Yu, and Yeh-Ching Chung, "Efficient Compositing Methods for the Sort-Last-Sparse Parallel Volume Rendering System on Distributed Memory Multicomputers," *J. Supercomput.*, vol. 18, no. 2, pp. 201–220, Feb. 2001.

[186] Hongfeng Yu, Chaoli Wang, Ray W. Grout, Jacqueline H. Chen, and Kwan-Liu Ma, "In Situ Visualization for Large-Scale Combustion Simulations," *IEEE Comput. Graph. Appl.*, vol. 30, no. 3, pp. 45–57, May 2010.

[187] Jiawan Zhang, Jizhou Sun, Zhou Jin, Yi Zhang, and Qi Zhai, "Survey of Parallel and Distributed Volume Rendering: Revisited," in *Proceedings of the 2005 International Conference on Computational Science and Its Applications - Volume Part III*, Berlin, Heidelberg, 2005, ICCSA'05, pp. 435–444, Springer-Verlag.

[188] Jianlong Zhou, Manfred Hinz, and Klaus D Tönnies, "Focal Region-Guided Feature-Based Volume Rendering 2 : Related Work," *Data Processing*, vol. ill, pp. 3–6, 2002.

[189] Tianshu Zhou, Jim X. Chen, and Mark Pullen, "Accurate Depth of Field Simulation in Real Time," *Computer Graphics Forum*, vol. 26, no. 1, pp. 15–23, Mar. 2007.

[190] Brian Cabral, Nancy Cam, and Jim Foran, "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware," in *Proceedings of the 1994 Symposium on Volume Visualization*, New York, NY, USA, 1994, VVS '94, pp. 91–98, ACM.