A Survey of GPU-Based Large-Scale Volume Visualization

Eurographics Conference on Visualization (EuroVis) (2014)
R. Borgo, R. Maciejewski, and I. Viola (Editors)

A Survey of GPU-Based Large-Scale Volume Visualization

Johanna Beyer[1], Markus Hadwiger[2], Hanspeter Pfister[1]

[1]Harvard University, USA
[2]King Abdullah University of Science and Technology, Saudi Arabia

Abstract

This survey gives an overview of the current state of the art in GPU techniques for interactive large-scale volume visualization. Modern techniques in this field have brought about a sea change in how interactive visualization and analysis of giga-, tera-, and petabytes of volume data can be enabled on GPUs. In addition to combining the parallel processing power of GPUs with out-of-core methods and data streaming, a major enabler for interactivity is making both the computational and the visualization effort proportional to the amount and resolution of data that is actually visible on screen, i.e., "output-sensitive" algorithms and system designs. This leads to recent output-sensitive approaches that are "ray-guided," "visualization-driven," or "display-aware." In this survey, we focus on these characteristics and propose a new categorization of GPU-based large-scale volume visualization techniques based on the notions of actual output-resolution visibility *and the current* working set *of volume bricks—the current subset of data that is minimally required to produce an output image of the desired display resolution. For our purposes here, we view parallel (distributed) visualization using clusters as an orthogonal set of techniques that we do not discuss in detail but that can be used in conjunction with what we discuss in this survey.*

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms

1. Introduction

Visualizing volumetric data plays a crucial role in scientific visualization and is an important tool in many domain sciences such as medicine, biology and the life sciences, physics, and engineering. The developments in GPU technology over the last two decades, and the resulting vast parallel processing power, have enabled compute-intensive operations such as ray-casting of large volumes at interactive rates. However, in order to deal with the ever-increasing resolution and size of today's volume data, it is crucial to use highly scalable visualization algorithms, data structures, and architectures in order to circumvent the restrictions imposed by the limited amount of on-board GPU memory.

Recent advances in high-resolution image and volume acquisition, as well as computational advances in simulation, have led to an explosion of the amount of data that must be visualized and analyzed. For example, high-throughput electron microscopy can produce volumes of scanned brain tissue at a rate above 10-40 megapixels per second [BLK*11], with a pixel resolution of 3-5 nm. Such an acquisition process produces almost a terabyte of raw data per day. For the next couple of years it is predicted that new multi-beam electron microscopes will further increase the data acquisition rate by two orders of magnitude [Hel13, ML13]. This trend of acquiring and computing more and more data at a rapidly increasing pace ("Big Data") will continue in the future [BCH12]. This naturally poses significant challenges to interactive visualization and analysis. For example, many established algorithms and frameworks for volume visualization do not scale well beyond a few gigabytes, and this problem cannot easily be solved by simply adding more computing power or disk space. These challenges require research on novel techniques for data visualization, processing, storage, and I/O that scale to extreme-scale data [MWY*09, AAM*11, BCH12].

Today's GPUs are very powerful parallel processors that enable performing compute-intensive operations such as ray-casting at interactive rates. However, the memory sizes available to GPUs are not increasing at the same rate as the amount of raw data. In recent years, several GPU-based methods have been developed that employ out-of-core methods and data streaming to enable the interactive visualization of giga-, tera-, and petabytes of volume data. The crucial property that enables these methods to scale to extreme-scale data is their *output-sensitivity*, i.e., that they make both the computational and the visualization effort proportional to the amount of data that is actually visible on screen (i.e., the output), instead of being proportional to the full amount

of input data. In graphics, the focus of most early work on output-sensitive algorithms was visibility determination of geometry (e.g., [SO92, GKM93, ZMHH97]).

An early work in output-sensitive visualization on GPUs was dealing with 3D line integral convolution (LIC) volumes of flow fields [FW08]. In the context of large-scale volume visualization, output-sensitive approaches are often referred to as being *ray-guided* (e.g., [CNLE09, Eng11, FSK13]) or *visualization-driven* (e.g., [HBJP12, BHAA*13]). These are the two terms that we will use most in this survey.

We use the term *visualization-driven* in a more general and inclusive way, i.e., these methods are not necessarily bound to ray-casting (which is implied by "ray-guided"), and they can encompass all computation and processing of data in addition to rendering. In principle, the visual output can "drive" the entire visualization pipeline—including on-demand processing of data—all the way back to the raw data acquisition stage [HBJP12, BHAA*13]. This would then yield a fully *visualization-driven pipeline*. However, to a large extent these terms can be used interchangeably.

Another set of output-sensitive techniques are *display-aware* multi-resolution approaches (e.g., [JST*10, JJY*11, HSB*12]). The main focus of these techniques is usually output-sensitive computation (such as image processing) rather than visualization, although they are also guided by the actual display resolution and therefore the visual output.

Ray-guided and visualization-driven visualization techniques are clearly inspired by earlier approaches for occlusion culling (e.g., [ZMHH97, LMK03]) and level of detail (e.g., [LHJ99, WWH*00]). However, they have a much stronger emphasis on leveraging *actual output-resolution visibility* for data management, caching, and streaming—in addition to the traditional goals of faster rendering and anti-aliasing. Very importantly, actual visibility is determined on-the-fly during visualization, directly on the GPU.

1.1. Survey Scope

This survey focuses on major scalability properties of volume visualization techniques, reviews earlier GPU volume renderers, and then discusses modern ray-guided and visualization-driven approaches and how they relate to and extend the standard visualization pipeline (see Figure 1). Large-scale GPU volume rendering can be seen as being in the intersection of volume visualization and high performance computing. General introductions to these two topics are given in books on real-time volume graphics [EHK*06] and high performance visualization [BCH12], respectively.

We mostly focus on techniques for stand-alone workstations with standard graphics hardware. We see the other core topics of high performance visualization (i.e., parallel rendering on CPU/GPU clusters, distributed visualization frameworks, and remote rendering) as an orthogonal set of techniques that can be used in combination with modern ray-guided, visualization-driven, and display-aware techniques as discussed here. Therefore, for more details on parallel visualization we refer the reader to previous surveys in this area [Wit98, BSS00, ZSJ*05]. Nonetheless, where parallel or distributed rendering methods do directly relate to our course of discussion we have added them to our exposition.

We focus on volume rendering of regular grids and mostly review methods for scalar data and a single time step. However, the principles of the discussed scalable methods are general enough that they also apply to multi-variate, multi-modal, or time series data. For a more in-depth discussion of the visualization and visual analysis of multi-faceted scientific data we refer the reader to a recent comprehensive survey [KH13]. Other related recent surveys can be found on the topics of compression for GPU-based volume rendering [RGG*13], and massive model visualization [KMS*06].

1.2. Survey Structure

This survey gives an overview of the current state of the art in large-scale GPU volume visualization. Starting from the standard visualization pipeline in Section 2, we discuss required modifications and extensions to this pipeline to achieve scalability with respect to data size (see Figure 1).

We continue by examining general scalability issues and how they relate to and are used in volume visualization (Section 3). This includes scalable data structures as well as data layout and compression for efficient data access on disk (Section 3.1). Next, we discuss different approaches for partitioning data and/or work to achieve scalable performance, from potentially in-core domain decomposition to out-of-core approaches (Section 3.2), before describing different ways to reduce the computational load, focusing on on-demand processing, streaming, and in-situ visualization approaches (Section 3.3).

Section 4 discusses recent advances in large-scale volume rendering in depth, starting with a review of traditional GPU volume rendering techniques and their limitations.

We focus on the characteristics of recent ray-guided, visualization-driven, and display-aware techniques (Section 4.1). To reflect and emphasize these recent advances, we propose a new categorization of GPU-based large-scale volume visualization techniques (Table 3) based on the notion of the active working set—the current subset of data that is minimally required to produce an output image of the desired display resolution.

We discuss methods for determining the working set, i.e., culling (Section 4.2), GPU data structures for storing the working set (Section 4.3), and the actual ray-casting methods for rendering the working set (Section 4.4).

Finally, we review the major challenges and current limitations and give an outlook on future trends and open problems in large-scale GPU volume visualization (Section 5).

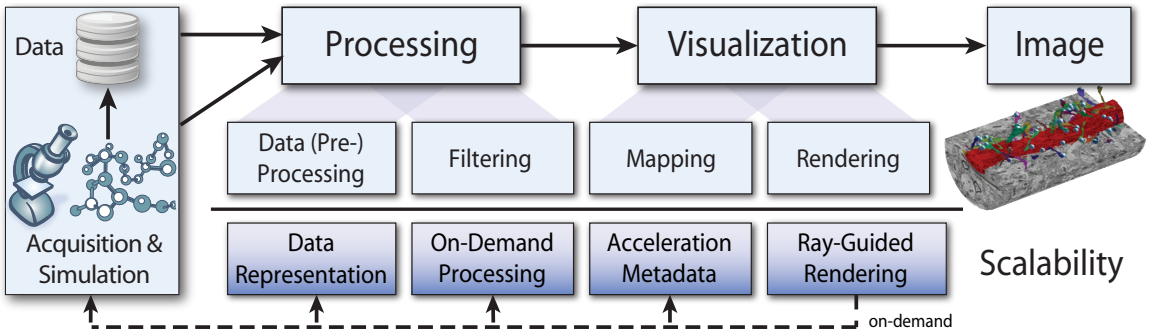

Figure 1: ***The visualization pipeline for large-scale visualization.*** *Data are generated on the left (either through acquisition/measurement or through computation/simulation) and then pass through a sequence of stages that culminate in the desired output image. The related high-level aspects with respect to scalability of interactive volume rendering are highlighted in the bottom row. A* ray-guided *or* visualization-driven *approach can drive earlier pipeline stages so that only what is required by (visible in) the output image is actually loaded or computed. In a fully visualization-driven pipeline, this approach can be carried through from rendering (determining visibility) on the right all the way back to data acquisition/simulation on the left.*

2. Fundamentals

We first introduce a few basic concepts and give a conceptual overview of the visualization pipeline with respect to large-scale volume visualization.

2.1. Basic Concepts

Large-scale visualization. In the context of this survey, large-scale visualization deals with volume data that do not completely fit into memory. In our case, the most important memory type is GPU on-board memory, but scalability must be achieved throughout the entire memory hierarchy. Most importantly, large-scale volume data cannot be handled directly by volume visualization techniques that assume that the entire volume is resident in memory in one piece.

Bethel et al. [BCH12] (Chapter 2) define large data based on three criteria: They are too big to be processed: (1) in their entirety, (2) all at one time, and (3) exceed the available memory. Scalable visualization methods and architectures tackle either one or a combination of these criteria.

Scalability. In contrast to parallel/distributed visualization, where a major focus is on *strong* vs. *weak* scaling [CPA*10], we define scalability in terms of *output-sensitivity* [SO92]. Our focus are algorithms, approaches, and architectures that scale to large data by making the computation and visualization effort proportional to both the *visible* data on screen and the actual *screen resolution*. If the required size of the *working set* of data is independent of the original data size, we say that an approach is *scalable* in this sense.

Scalability issues. Based on the notion of large data, the main scalability issues for volume rendering deal with questions on how to represent data, how to split up the work and/or data to make it more tractable, and how to reduce the amount of work and/or data that has to be handled. Table 1 lists these main issues and the general methods that are used in large-scale visualization to handle them.

Acceleration techniques vs. data size. A common source of confusion when discussing techniques for scalable volume rendering is the real goal of a specific optimization technique. While many of the techniques discussed in this survey were originally proposed as performance optimizations, they can also be adapted to handle large data sizes. A well-known example of this are octrees. While octrees are often used in geometry rendering to speed up view frustum culling (via hierarchical/recursive culling), an important goal of using octrees in volume rendering is to enable adaptive level of detail [WWH*00], in addition to enabling empty space skipping. This "dual" purpose of many scalable data structures and algorithms is an important issue to keep in mind.

Output-sensitive algorithms. The original focus of output-sensitive algorithms [SO92] was making their *running time* dependent on the size of the *output* instead of the size of the *input*. While this scalability in terms of running time is of course also important in our context, for the work that we discuss here, it is even more important to consider the dependence on output "data size" vs. input data size, using the concept of the *working set* as described above.

Ray-guided and visualization-driven architectures. In line with the concepts outlined above, these types of architectures focus most of all on data management (processing, streaming, caching) rather than only on rendering. While ray-casting intrinsically could be called "ray-guided," this by itself is not very meaningful. The difference to standard ray-casting first arises from how and which data are streamed into GPU memory, i.e., *ray-guided streaming* of volume data [CNLE09]. Again considering the working set, a ray-guided approach determines the working set of volume bricks *via* ray-casting. That is, the working set comprises the bricks that are intersected during ray traversal. It is common

scalability issues	scalable methods	section
data representation	multi-res. data structures	Sec. 3.1.2
	data layout, compression	Sec. 3.1.3
work/data partitioning	in-core/out-of-core	Sec. 3.2.2
	parallel/distributed	Sec. 3.2.3
work/data reduction	pre-processing	Sec. 3.3.1
	on-demand processing	Sec. 3.3.2
	streaming	Sec. 3.3.3
	in-situ visualization	Sec. 3.3.4
	query-based visualization	Sec. 3.3.5

Table 1: ***Scalability considerations in large-scale volume visualization.*** *Scalability issues, the corresponding methods to tackle them, and where they are covered in this survey.*

to determine the desired level of detail, i.e., the (locally) required volume resolution, during ray-casting as well.

In this way, data streaming is guided by the *actual* visibility of data in the output image. This is in contrast to the approximate/conservative visibility obtained by all common occlusion culling approaches. As described in the introduction, *visualization-driven* architectures generalize these concepts further to ultimately drive the entire visualization pipeline by actual on-screen visibility [HBJP12, BHAA*13].

2.2. Large-Scale Visualization Pipeline

A common abstraction used by visualization frameworks is the *visualization pipeline* [Mor13]. In essence, the visualization pipeline is a data flow network where nodes or modules are connected in a directed graph that depicts the data flow throughout the system (see Figure 1). After data acquisition or generation through computation/simulation, the first stage usually consists of some kind of data processing, which can include many sub-tasks from data *pre-processing* (e.g., computing a multi-resolution representation) to *filtering*. The second half of the pipeline comprises the actual visualization, including *visualization mapping* and *rendering*.

For large-scale rendering, all the stages in this pipeline have to be scalable (i.e., in our context: output-sensitive), or they will become the bottleneck for the entire application. The bottom part of Figure 1 shows the main techniques employed by state-of-the-art visualization-driven pipelines to achieve this scalability: Multi-resolution and compact data representations, on-demand processing based on the visible subset currently in view, acceleration data (e.g., for faster ray traversal or empty space skipping), and ray-guided rendering with dynamic ray traversal.

Table 1 gives an overview of the most important scalability aspects of large-scale visualization frameworks that we will use later. Actual scalability also depends on how dynamically and accurately the working set is determined, how volumes are represented, and how ray traversal is performed. We discuss individual visualization methods in Section 4.

3. Basic Scalability Techniques

This section introduces the main considerations and techniques for designing scalable volume visualization architectures in general terms. In real-world applications, these strategies for handling and rendering large data often have to be combined to achieve interactive performance and high-quality images.

For future ultra-scale visualization and exa-scale computing [ALN*08, SBH*08, MWY*09, AAM*11, Mor12] it is essential that each step of the visualization pipeline is fully scalable.

3.1. Data Representation and Storage

Efficient data representation is a key requirement for scalable volume rendering. Scalable data structures should be compact in memory (and disk storage), while still being efficient to use and modify. Table 2 lists common related data structures and their scalability aspects. Additional GPU representations of these data structures, as they are used for rendering, are discussed in Section 4.4.

3.1.1. Bricking

Bricking is an object space decomposition method that subdivides the volume into smaller, box-shaped sub-volumes, or *bricks*. Commonly, all bricks have the same size in voxels (e.g., 32^3 or 256^3 voxels per brick). Volumes that are not a multiple of the basic brick size are padded accordingly. Bricking facilitates out-of-core approaches because individual bricks can be loaded and rendered as required, without having to load/stream the volume in its entirety.

Bricked data usually require special handling of brick boundaries. Operations where neighboring voxels are required (e.g., GPU texture filtering, gradients) usually return incorrect results at brick boundaries, because the neighboring voxels are not readily available. The correct voxels can be fetched from the neighboring bricks [Lju06a], which is costly. More commonly, so-called *ghost voxels* [ILC10] are employed, which are duplicated voxels at the brick boundaries that enable straightforward, correct filtering. The use of ghost voxels is the standard approach in most bricked ray-casters [BHWB07, FK10]. Ghost voxels are usually stored with each brick on disk, but they can also be computed on-the-fly in a streaming fashion [ILC10].

The recent OpenGL extension for virtual texturing (`GL_ARB_sparse_texture`) includes hardware support for texture filtering across brick boundaries and thus alleviates the need for ghost voxels.

Choosing the optimal brick size depends on several criteria and has been studied in the literature [HBJP12, FSK13]. Small bricks support fine-grained culling, which results in smaller working sets. However, the ghost voxel overhead

Data Structure	Acceleration	Out-of-Core	Multi-Resolution
mipmaps	no [except level of detail]	clipmaps [TMJ98]	yes
octrees / kd-trees	hierarchical traversal/culling	working set (subtree)	yes
uniform grids/bricking	(linear) culling of bricks	working set (bricks from grid)	no
hierarchical grids/bricking	(hierarchical) culling of bricks	working set (bricks from hierarchy)	yes

Table 2: ***Scalable data structures for volume visualization.*** *Our categorization is based on their support for acceleration (skipping, culling), out-of-core processing/rendering, and support for multi-resolution rendering (i.e., adaptive level of detail).*

grows for smaller bricks, and the total number of bricks increases as well. The latter makes a multi-pass rendering approach where each brick is rendered individually infeasible.

Typically, traditional multi-pass out-of-core volume renderers use relatively large bricks (e.g., 128^3 or 256^3) to reduce the number of required render passes. In contrast, modern single-pass ray-casters use smaller bricks (e.g., 32^3), or a hybrid approach where small bricks are used for rendering and larger bricks are used for storage on disk [HBJP12, FSK13]. For 2D data acquisition modalities such as microscopy, hybrid 2D/3D tiling/bricking strategies have also been employed successfully, for example via on-demand computation of 3D bricks from pre-computed 2D mipmap tiles during visualization [HBJP12, BHAA*13].

3.1.2. Multi-Resolution Hierarchies

One of the main benefits of multi-resolution hierarchies for rendering large data is that they allow sampling the data from a resolution level that is adapted to the current screen resolution or desired level of detail. This reduces the amount of data to be accessed and also avoids aliasing artifacts due to undersampling.

Trees (octrees, kd-trees). Octrees [WWH*00, Kno06] and kd-trees [FCS*10] are very common 3D multi-resolution data structures for direct volume rendering. They allow efficient traversal and directly support hierarchical empty space skipping. Traditional tree-based volume renderers employ a multi-pass rendering approach where one brick (one tree node) is rendered per rendering pass. Despite the hierarchical nature of these data structures, many early approaches assume that the entire volume fits into memory [LHJ99, WWH*00, BNS01]. Modern GPU approaches support traversing octrees directly on the GPU [GMG08, CNLE09, CN09, RTW13], which is usually accomplished via standard traversal algorithms from the ray-tracing literature [AW87, FS05, HSHH07, PGS*07, HL09].

In recent years, *sparse voxel octrees* (SVOs) have gained a lot of attention in the graphics and gaming industry [LK10a, LK10b]. Several methods for rendering large and complex voxelized 3D models use SVO data structures for efficient rendering [GM05, RÖ9, HN12, Mus13].

Mipmaps are a standard multi-resolution pyramid representation that is very common in texture mapping [Wil83]. Mipmaps are supported by virtually all GPU texture units. Clipmaps [TMJ98] are virtualized mipmaps of arbitrary size. They assume a moving window (like in terrain rendering) that looks at a small sub-rectangle of the data and use a toroidal updating scheme for texels in the current view.

Hierarchical grids with bricking. Another type of multi-resolution pyramids are hierarchical grids where each resolution level of the data is bricked individually. These grids have become a powerful alternative to octrees in recent ray-guided volume visualization approaches [HBJP12, FSK13]. The basic approach can be viewed as bricking each level of a mipmap individually. However, more flexible systems do not use hardware mipmaps and therefore allow varying down-sampling ratios between resolution levels [HBJP12]—e.g., for anisotropic data—which is not possible with mipmaps.

Since there is no tree structure in such a grid type, no tree traversal is necessary during rendering. Rather, the entire grid hierarchy is viewed as a huge virtual address space (a *virtual texture*), where any voxel corresponding to data of any resolution can be accessed directly via *address translation* from *virtual* to *physical* addresses [vW09, BHL*11, OVS12]. On GPUs, this address translation can be performed via GPU "page tables," which is also possible in a *multi-level* way for extremely large data [HBJP12] (see Section 4.4.1). As in the case of bricking with uniform grids, interpolation between bricks has to be handled carefully. Especially transitions between different resolution levels can introduce visual artifacts, and several methods have been introduced that deal with correct interpolation [Lju06a, Lju06b, BHMF08].

Wavelet representations. Muraki [Mur93] first introduced wavelet transforms for volume rendering. Subsequent methods such as Guthe et al. [GGSe*02, GS04] compute a hierarchical wavelet representation in a pre-process and decompress the bricks required for rendering

Other representations. Younesy et al. [YMC06] have proposed improving the visual quality of multi-resolution volume rendering by approximating the voxel data distribution by its mean and variance at each level of detail. The recently introduced *sparse pdf maps* represent the data distribution more accurately, allowing for the accurate, anti-aliased evaluation of non-linear image operators on gigapixel images [HSB*12]. The corresponding data structure is very similar to standard mipmaps in terms of storage and access.

3.1.3. Data Layout and Compression

Data layout. To efficiently access data on disk, data layout and access are often optimized. In general, reading small bits of data at randomly scattered positions is a lot more inefficient than reading larger chunks in a continuous layout. Therefore, locality-preserving data access patterns such as space filling curves, e.g., Morton (z-) order [Mor66] are often used in time-critical visualization frameworks [SSJ*11]. A nice feature of the Morton/z-order curve is that by adjusting the sampling stride along the curve, samples can be restricted to certain resolution levels. Pascucci and Frank [PF02] describe a system for progressive data access that streams in missing data points for higher resolutions. With the most recent solid state drives (SSDs), however, trade-offs might be different in practice [FSK13].

Data compression. Another major related field is data compression, for disk storage as well as for the later stages of the visualization pipeline. We refer to the recent comprehensive survey by Rodriguez et al. [RGG*13] for an in-depth discussion of the literature on volume compression and *compression-domain* volume rendering.

3.2. Work/Data Partitioning

A crucial technique for handling large data is to *partition* or *decompose* data into smaller parts (e.g., sub-volumes). This is essentially a *divide and conquer* strategy, i.e., breaking down the problem into several sub-problems until they become easier to solve. Partitioning the data and/or work can alleviate memory constraints, complexity, and allow parallelization of the computational task. In the context of visualization, this includes ideas like domain decomposition (i.e., object-space and image-space decompositions), but also entails parallel and distributed visualization approaches.

3.2.1. Domain Decompositions

Object-space (data domain) decomposition is usually done by using bricking with or without a multi-resolution representation, as described in Sections 3.1.1 and 3.1.2, respectively. Object-space decompositions are view-independent and facilitate scalability with respect to data size by storing and handling data subsets separately.

Image-space (image domain) decomposition subdivides the output image plane (the viewport) and renders the resulting image tiles independently. A basic example of this approach is ray-casting (which is an *image-order* approach), where conceptually each pixel is processed independently. In practice, several rays (e.g., a rectangular image tile) are processed together in some sense. For example, rendering each image tile in a single rendering pass, or assigning each tile to a different rendering node. Another example is rendering on a large display wall, where each individual screen is assigned to a different rendering node.

3.2.2. Out-Of-Core Techniques

Unless when dealing with data that is small enough to fit into memory ("in core") in its entirety, one always has to partition the data and/or computation in a way that makes it possible to process subsets of the data independently. This enables out-of-core processing and can be applied at all stages of the visualization pipeline [SCC*02, KMS*06]. Different levels of out-of-core processing exist, depending on where the computation is performed and where the data is residing (either on the GPU, CPU, hard-disk, or network storage).

Out-of-core methods include algorithms that focus on accessing [PF02] and prefetching [CKS03] data, creating on-the-fly ghost data for bricked representations [ILC10], and methods for computing multi-resolution hierarchies [HBJP12] or other processing tasks such as segmentation [FK05], PDE solvers [SSJ*11], image registration and alignment [JST*10], or level set computation [LKHW04].

Silva et al. [SCC*02] give a comprehensive overview of out-of-core methods for visualization and graphics.

3.2.3. Parallel and Distributed Rendering

High-performance visualization often depends on distributed approaches that split the rendering of a data set between several nodes of a cluster. The difference can be defined such that *parallel* visualization approaches run on a single large parallel platform, whereas *distributed* approaches run on a heterogeneous network of computers. Molnar et al. [MCE*94] propose a classification of parallel renderers into *sort-first*, *sort-middle*, and *sort-last*. In the context of large data volume rendering, *sort-last* approaches are very popular. In this context, this term refers to bricking the data and making each node responsible for rendering one or several bricks before final image compositing. In contrast, sort-first approaches subdivide the viewport and assign render nodes to individual image tiles. Neumann [Neu94] examines the communication costs for different parallel volume rendering algorithms.

Conceptually, all or any parts of the visualization pipeline can be run as a distributed or parallel system. Recent developments in this field are promising trends towards exascale visualization. However, covering the plethora of distributed and parallel volume visualization approaches is out of scope of this survey. The interested reader is referred to [Wit98, BSS00, ZSJ*05] and [BCH12] (Chapter 3) for in-depth surveys on this topic.

3.3. Work/Data Reduction

Reducing the amount of data that has to be processed or rendered is a major strategy for dealing with large data. Techniques for data reduction cover a broad scope, ranging from multi-resolution data representations and sub-sampling to

more advanced filtering and abstraction techniques. A distinction has to be made between data reduction for storage (e.g., compression) that tries to reduce disk or in-memory size, and data reduction for rendering. The latter encompasses visualization-driven and display-aware rendering approaches as well as more general methods such as on-demand processing and query-based visualization.

3.3.1. Pre-Processing

Running computationally expensive or time-consuming computations as a pre-process to compute acceleration metadata or pre-cache data can often dramatically reduce the computation costs during rendering. Typical examples include pre-computing a multi-resolution hierarchy of the data that is used to reduce the amount of data needed for rendering. On the other hand, processing data interactively during rendering can reduce the required disk space [BCH12] (Chapter 9), and enables on-demand processing, which in turn can reduce the amount of data that needs processing.

3.3.2. On-Demand Processing

On-demand strategies determine at run time which parts of the data need to be processed, thereby eliminating preprocessing times and limiting the amount of data that needs to be handled. For example, ray-guided and visualization-driven volume rendering systems only request volume bricks to be loaded that are necessary for rendering the current view [CNLE09, HBJP12, FSK13]. Data that is not visible is never rendered, processed, or even loaded from disk.

Other examples for on-the-fly processing for volume visualization target interactive filtering and segmentation. For example, Jeong et al. [JBH*09] have presented a system where they perform on-the-fly noise removal and edge enhancement during volume rendering only for the currently visible part of the volume. Additionally, they perform an interactive active-ribbon segmentation on a dynamically selected subset of the data.

3.3.3. Streaming

In streaming approaches, data are processed as they become available (i.e., are streamed in). Streaming techniques are closely related to on-demand processing. However, where the latter usually consists of a *pull model* (i.e., data is requested by a process), streaming can be a pull or a *push model* (i.e., new data is pushed to the next processing step).

Streaming also facilitates circumventing the need for the entire data set to be available before the visualization starts and allows rendering of incomplete data [SCC*02]. Hadwiger et al. [HBJP12] have described a system for streaming extreme-scale electron microscopy data for interactive visualization. This system has later been extended to include on-the-fly registration and multi-volume visualization of segmented data [BHAA*13]. Further streaming-based visualization frameworks include the dataflow visualization system presented by Vo et al. [VOS*10], which is built on top of VTK and implements a push and pull model.

3.3.4. In-Situ Visualization

Traditionally, visualization is performed after all data have been generated—either by measurement or simulation—and have been written to disk. In-situ visualization, on the other hand, runs simultaneously to the on-going simulation (e.g., on the same supercomputer or cluster: *in situ*—in place), with the aim of reducing the amount of data that needs to be transferred and stored on disk [BCH12] (Chapter 9).

To avoid slowing down the primary simulation, *in-transit* visualization accesses only "staging" nodes of a simulation cluster. The goal of these nodes is to hide the latency of disk storage from the main simulation by handling data buffering and I/O [MOM*11].

In-situ and in-transit visualization have been identified as being crucial for future extreme-scale computing [MWY*09, AAM*11, KAL*11, Mor12]. Furthermore, when the visualization process is tightly coupled or integrated into the simulation, these approaches can be leveraged for *computational steering*, where simulation parameters are changed based on the visualization [PJ95, TTRU*06]. Yu et al. [YWG*10] present a complete case study of in-situ visualization for a petascale combustion simulation. Tikhonova et al. [TYC*11] take a different approach by generating a compact intermediate representation of large volume data that enables fast approximate rendering for preview and in-situ setups.

3.3.5. Query-based Visualization

Query-driven visualization uses selection as the main means to reducing the amount of data that needs to be processed [BCH12] (Chapter 7). Prominent techniques are dynamic queries [AWS92], high-dimensional brushing and linking [MW95], and interactive visual queries [DKR97]. Shneiderman [Shn94] gives an introduction to dynamic queries for visual analysis and information seeking.

The DEX framework [SSWB05] focuses on query-driven scientific visualization of large data sets using bitmap indexing to quickly query data. Recently, approaches for query-based volume visualization have been introduced in the context of neuroscience [BvG*09, BAaK*13], with the goal to analyze the connectivity between individual neurons in electron microscopy volumes. The ConnectomeExplorer framework [BAaK*13] implements visual queries on top of a large-scale, visualization-driven system.

4. Scalable Volume Rendering Techniques

In this section we categorize and discuss the individual literature in GPU-based large-scale volume rendering. We start

working set determination	full volume	basic culling (global, view frustum)		ray-guided / visualization-driven
volume data representation	linear (non-bricked) volume storage [CN93] [CCF94] [WE98] [RSEB*00] [HBH03] [LMK03]† [RGW*03] [KW03] [SSKE05] [BG05] [MHS08] [KGB*09]† [MRH10]	single-resolution grid [HSSB05] [BHWB07] grid with octree per brick [RV06]	octree [LHJ99] [WWH*00] [GGSe*02] [GS04] [PHKH04] [HFK05] kd-tree [FK10] multi-resolution grid [Lju06a] [BHMF08] [JBH*09]	octree [GMG08]‡ [CNLE09] [Eng11] [RTW13] multi-resolution grid [HBJP12] [BAaK*13] [FSK13]
rendering (ray traversal)	texture slicing [CN93] [CCF94] [WE98] [RSEB*00] [HBH03] [LMK03]† non-bricked ray-casting (multi-pass) [RGW*03] [KW03] (single-pass) [SSKE05] [BG05] [MHS08] [KGB*09]† [MRH10]	CPU octree traversal (multi-pass) [LHJ99] [WWH*00] [GGSe*02] [GS04] [PHKH04] [HFK05] [RV06] CPU kd-tree traversal (multi-pass) [FK10] bricked/virtual texture ray-casting (single-pass) [HSSB05] [Lju06a] [BHWB07] [BHMF08] [JBH*09]		GPU octree traversal (single-pass) [GMG08]‡ [CNLE09] [Eng11] [RTW13] multi-level virtual texture ray-casting (single-pass) [HBJP12] [BAaK*13] [FSK13]
scalability	low	medium		high

Table 3: ***Categorization of GPU-based volume visualization techniques*** *based on the type of* working set determination *mechanism and the resulting* scalability *in terms of data size, as well as according to the volume data representation employed, and the actual rendering technique (type of ray traversal; except in the case of texture slicing).* † *[LMK03, KGB*09] perform culling for empty space skipping, but store the entire volume in linear (non-bricked) form.* ‡ *[GMG08] is not fully ray-guided, but utilizes interleaved occlusion queries with similar goals (see the text).*

with an overview of "traditional" GPU-based volume rendering techniques, before we go into details on "modern" ray-guided and visualization-driven techniques.

Categorization (Table 3). We categorize GPU-based volume rendering approaches with respect to their scalability properties by using the central notion of the *working set*—the subset of volume bricks that is required for rendering a given view. Using the concept of working set, our categorization distinguishes different approaches according to:

1. How the working set is determined.
2. How the working set is stored (represented) on the GPU.
3. How the working set is used (accessed) during rendering.

We elaborate on these categories below in (1) Section 4.2, (2) Section 4.3, and (3) Section 4.4.

We also categorize the resulting *scalability* (low, medium, high), where only "high" scalability means full output-sensitivity and thus independence of the input volume size.

The properties of different volume rendering approaches—and the resulting scalability—vary greatly between what we refer to as "traditional" approaches (corresponding to "low" and "medium" scalability in Table 3), and "modern" ray-guided approaches (corresponding to "high" scalability in Table 3).

A key feature of modern ray-guided and visualization-driven volume renderers is that they make full use of recent developments in GPU programmability. They usually include a read-back mechanism to update the current working set, and traverse a multi-resolution hierarchy dynamically on the GPU. This flexibility was not possible on earlier GPUs and is crucial for determining an accurate working set.

4.1. GPU-Based Volume Rendering

GPUs have, over the last two decades, become very versatile and powerful parallel processors, succeeding the fixed-function pipelines of earlier graphics accelerators. General purpose computing on GPUs (GPGPU)—now also called GPU Compute—leverages GPUs for non-graphics related and compute-intensive computations [OLG*07], such as simulations or general linear algebra problems. Increased programmability has been made possible by APIs like the OpenGL Shading Language (GLSL) [Ros06] and CUDA [NVI13].

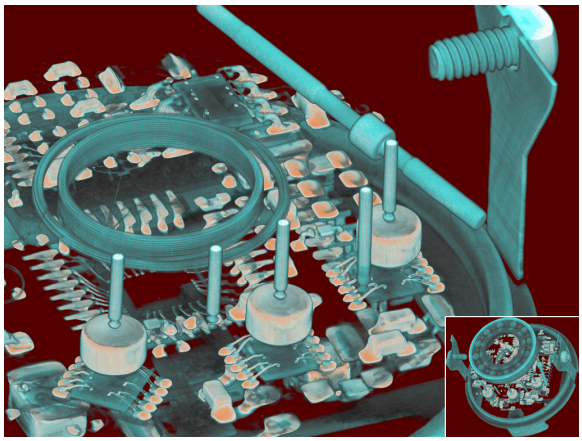

Figure 2: *Rendering a multi-gigabyte CT data set (as used in [Eng11]) at different resolution levels using a ray-guided rendering approach. Data courtesy of Siemens Healthcare, Components and Vacuum Technology, Imaging Solutions. Data was reconstructed by the Siemens OEM reconstruction API CERA TXR (Theoretically Exact Reconstruction).*

However, GPU on-board memory sizes are much more limited than those of CPUs. Therefore, large-scale volume rendering on GPUs requires careful algorithm design, memory management, and the use of out-of-core approaches.

4.1.1. Traditional GPU-Based Volume Rendering

Before discussing current state-of-the-art ray-guided volume renderers, we review traditional GPU volume rendering approaches. We start with 2D and 3D texture slicing methods, before continuing with GPU ray-casting. This will give us the necessary context for categorizing and differentiating between the more traditional and the more modern approaches.

Texture slicing. The earliest GPU volume rendering approaches were based on texture mapping [Hec86] using 2D and 3D texture slicing [CN93, CCF94]. Westermann and Ertl [WE98] extended this approach to support arbitrary clipping geometries and shaded iso-surface rendering. For correct tri-linear interpolation between slices, Rezk-Salama et al. [RSEB*00] made use of multi-texturing. Hadwiger et al. [HBH03] described how to efficiently render segmented volumes on GPUs and how to perform two-level volume rendering on GPUs, where each labeled object can be rendered with a different render mode and transfer function. This approach was later extended to ray-casting of multiple segmented volumes [BHWB07]. Engel et al. [ESE00] were among the first to investigate remote visualization using hardware-accelerated rendering.

Texture slicing and parallel volume rendering. Texture slicing has been used in many distributed and parallel volume rendering systems [MHE01, CMC*06, MWMS07, EPMS09, FCS*10]. Magallon et al. [MHE01] used sort-last rendering on a cluster, where each cluster node renders one volume brick before doing parallel compositing for final image generation. For volume rendering on small to medium GPU clusters, Fogal et al. [FCS*10] introduced a load-balanced sort-last renderer integrated into VisIt [CBB*05], a parallel visualization and data analysis framework for large data sets. Moloney et al. [MWMS07] proposed a sort-first technique using eight GPUs, where the render costs per pixel are used for dynamic load balancing between nodes. They later extended their method to support early ray termination and volume shadowing [MAWM11]. Equalizer [EPMS09] is a GPU-friendly parallel rendering framework that supports both sort-first and sort-last approaches.

Texture slicing today. In general, the advantage of texture slicing-based volume renderers is that they have minimum hardware requirements. 2D texture slicing, for example, can be implemented in WebGL [CSK*11] and runs efficiently on mobile devices without 3D texture support. However, a disadvantage is that they often exhibit visual artifacts and less flexibility when compared to ray-casting methods.

Ray-casting. Röttger et al. [RGW*03] and Krüger and Westermann [KW03] were among the first to perform ray-casting on GPUs, using a multi-pass approach. Ray-casting is embarrassingly parallel and can be implemented on the GPU in a fragment shader or compute kernel, where each fragment or thread casts one ray through the volume. Ray-casting easily admits a wide variety of performance and quality enhancements such as empty space skipping and early ray termination. Hadwiger et al. [HSSB05] and Stegmaier et al. [SSKE05] were among the first to perform GPU ray-casting using a single-pass approach, taking advantage of dynamic looping and branching in then-recent GPUs. Proxy geometries for efficient empty space skipping can be based on bricks [HSSB05, SHN*06], spheres [LCD09], or occlusion frustums [MRH08].

Müller et al. [MSE06] used GPU ray-casting in a sort-last parallel rendering system. With the introduction of CUDA as a higher-level GPU programming language, CUDA-based ray-casters were introduced [MHS08, KGB*09, MRH10]. They make use of CUDA's thread/block architecture, and possibly shared memory model.

Large data. For rendering large data, several multi-resolution octree rendering methods have been proposed, most of them based on texture-slicing [LHJ99, WWH*00, GGSe*02, GS04, PHKH04]. Hong et al. [HFK05] used a min-max octree structure for ray-casting the Visible Human CT data set. To support volumes that are larger than GPU memory, bricked single-pass ray-casting can be used [HSSB05, BHWB07, JBH*09]. These techniques access volume bricks stored in a large brick cache (or brick pool) texture, which is similar to adaptive texture maps [KE02]. However, the brick cache is usually managed dynamically to accommodate transfer function changes. Ljung et al. [Lju06a] used a multi-resolution bricking structure and adaptive sampling in image- and object-space to

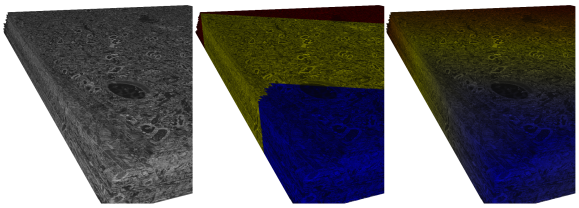

Figure 3: ***Per-sample LOD selection*** *as in [HBJP12]. Left: electron microscopy volume (90 GB). Middle and right: the LOD used for each sample is color-coded. Middle: discrete LOD for each sample (tri-linear interpolation). Right: fractional LOD for each sample, with interpolation between data of neighboring LODs ("quad-linear" interpolation).*

render large data. Beyer et al. [BHMF08] proposed a technique for correct interpolation between bricks of two different resolution levels.

A lot of research has focused on remote, parallel, or distributed visualization for rendering large data, which we cannot all cover here. For example, Prohaska et al. [PHKH04] used an octree approach to remotely render large remote micro-CT scans, while Wang et al. [WGL*05] proposed a wavelet-based time space partitioning tree for volume rendering of large time varying volumes but use a parallel CPU ray-caster on a PC cluster for rendering.

A different approach to dealing with large data was proposed by Turlington et al. [THM01], who introduced sliding thin slab (STS) visualization to limit the amount of data needed for any current view. Knoll et al. [KTW*11] optimized CPU ray-casting, achieving interactive rates using a bounding volume hierarchy (BVH) min/max acceleration structure and SIMD optimizations.

4.1.2. Ray-Guided Volume Rendering

Ray-guided and visualization-driven volume rendering approaches incorporate a feedback loop between the ray-caster and the culling mechanism, where the ray-caster itself writes out accurate information on missing bricks and brick usage. Thus, this type of culling mechanism determines an accurate working set directly on the GPU.

This information about the working set is then used to load missing data, and to determine which bricks can be evicted from the GPU cache because they are no longer needed. Additionally, rays automatically determine the (locally) required data resolution. This determination can be performed either on a per-sample basis [HBJP12] (see Figure 3), or on a per-brick basis [FSK13].

Gobbetti et al. [GMG08] were among the first to implement a volume ray-caster with stackless GPU octree traversal. They used occlusion queries to determine, load, and possibly refine visible nodes. This approach already has similar properties to later fully ray-guided approaches. However, it is strictly speaking not fully ray-guided, because culling of octree nodes is performed on the CPU based on the occlusion query information obtained from the GPU.

Crassin et al. [CN09] introduced the Gigavoxels system for GPU-based octree volume rendering with *ray-guided streaming* of volume data. Their system can also make use of an N^3 tree, as an alternative to an octree (which would be an N^3 tree with $N = 2$). The tree is traversed at run time using the kd-restart algorithm [FS05] and active tree nodes stored in a *node pool*. Actual voxel data are fetched from bricks stored in a *brick pool*. Each node stores a pointer to its child nodes in the node pool, and a pointer to the associated texture brick in the brick pool (see Figure 4). The focus of the Gigavoxels system is volume rendering for entertainment applications and as such it does not support dynamic transfer function changes [CNSE10]. The more recent CERA-TVR system [Eng11] targets scientific visualization applications and supports fully dynamic updates according to the transfer function in real time. It also uses the kd-restart algorithm for octree traversal. Reichl et al. [RTW13] also employ a similar ray-guided approach, but target large smooth particle hydrodynamics (SPH) simulations.

A different category of ray-guided volume renderers uses hierarchical grids with bricking, which are accessed via multi-level page tables instead of a tree structure. Hadwiger et al. [HBJP12] proposed such a multi-resolution virtual memory scheme based on a multi-level page table hierarchy (see Figure 5). This approach scales to petavoxel data and can also efficiently handle highly anisotropic data, which is very common in high-resolution electron microscopy volumes. They also compare their approach for volume traversal to standard octree traversal in terms of traversal complexity and cache access behavior, and illustrate the advantages of multi-level paging in terms of scaling to very large data.

Fogal et al. [FSK13] have performed an in-depth analysis of several aspects of ray-guided volume rendering.

4.2. Working Set Determination

Performing culling to determine the current working set of bricks is crucial for ray-casting large data at interactive frame rates. Originally, culling was introduced for geometry rendering, where view frustum and occlusion culling are used to limit the number of primitives that have to be rendered. Ideally, all *occluded* geometry should be culled before rendering.

4.2.1. View Frustum Culling

Removing primitives or volume bricks outside the current view frustum is the most basic form of culling. The first step of GPU ray-casting consists of computing the ray start points and end points (often via rasterization), which already prevents sampling the volume in areas that are outside the view

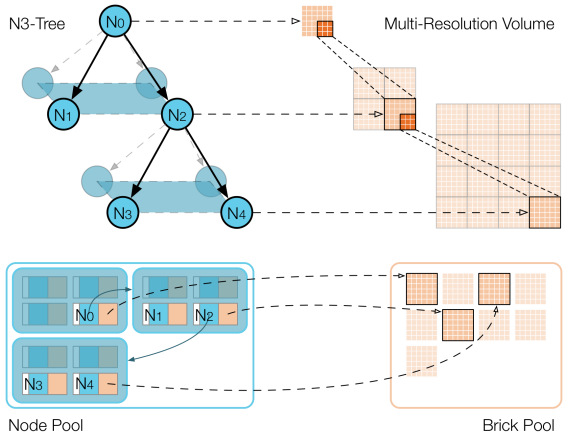

Figure 4: ***The Gigavoxels system*** *uses an N^3 tree structure with node and brick pools that store the set of active nodes and bricks, respectively.*

frustum. However, in order to prevent volume bricks outside the frustum from being downloaded to the GPU, the individual bricks have to be culled against the view frustum. Naturally, if a brick lies completely outside the current view frustum, it is not needed in GPU memory. Culling a view frustum against a bounding box, a bounding volume hierarchy, or a tree can be done very efficiently and has been studied extensively [AM00, AMHH08].

4.2.2. Global, Attribute-Based Culling

Another way to cull bricks in volume rendering is based on global properties like the current transfer function, iso value, or enabled segmented objects. Culling against the transfer function is usually done based on min/max computations for each brick [PSL*98, HSSB05, SHN*06]. The brick's min/max values are compared against the transfer function to determine if the brick is invisible (i.e., only contains values that are completely transparent in the transfer function). Invisible bricks are then culled. The downside of this approach is that it needs to be updated whenever the transfer function changes and usually needs pre-computed min/max values for each brick that have to be available at runtime for all bricks. A similar approach can be used for culling bricks against an iso-surface [PSL*98, HSSB05], or against enabled/disabled objects in segmented volume rendering [BHWB07].

4.2.3. Occlusion/Visibility Culling

Occlusion or visibility culling tries to cull primitives inside the view frustum that are occluded by other primitives. While this is easier for opaque geometry, in (transparent) volume rendering this process is more involved and often requires a multi-pass rendering approach.

Greene et al. [GKM93] introduce hierarchical z-buffer visibility. They use two hierarchical data structures, an octree in object space and a z-pyramid in image space to quickly reject invisible primitives in a hierarchical manner. Zhang et al. [ZMHH97] propose hierarchical occlusion maps (HOMs), where a set of occluders is rendered into a low-resolution occlusion map that is hierarchically downsampled and used to test primitives for occlusion before rendering them.

For volume visualization, Li et al. [LMK03] introduce occlusion clipping for texture-based volume rendering to skip rendering of occluded parts of the volume. Gao et al. [GHSK03] propose visibility culling in large-scale parallel volume rendering based on pre-computing a plenoptic opacity function per brick. Visibility culling based on temporal occlusion coherence has also been used for time-varying volume rendering [GSHK04]. The concept of occlusion culling has also been used in a parallel setting for sort-last rendering [MM10], by computing and propagating occlusion information across rendering nodes.

4.2.4. Ray-Guided Culling

Ray-guided culling approaches are different in the sense that they start with an empty working set. Only bricks that are actually visited during the ray-casting traversal step are requested and subsequently added to the working set of active bricks. Therefore, this approach implicitly culls all occluded bricks, as well as bricks outside the view frustum.

Gobbetti et al. [GMG08] use a mixture of traditional culling and ray-guided culling. They first perform culling on the CPU (using the transfer-function, iso value, and view frustum), but refine only those nodes of the octree that were marked as visible in the previous rendering pass. To determine if a node is visible they use occlusion queries to check the bounding box of a node against the depth of the last visited sample that was written out during ray-casting.

Crassin et al. [CN09] originally used multiple render targets to report which bricks were visited by the ray-caster over the course of several frames, exploiting spatial and temporal coherence. The same information was constructed in a more efficient way using CUDA in a later implementation [CNSE10].

Hadwiger et al. [HBJP12] divide the viewport into smaller tiles and use a GPU hash table per image tile to report a limited number of cache misses. Over the course of several frames, this ensures that all missing bricks are reported.

Fogal et al. [FSK13] use a similar approach built on lock-free hash tables.

4.3. Working Set Storage and Access

Efficient GPU data structures for storing the working set should be fast to access during ray traversal, and should also support efficient dynamic updates of the working set. Recent approaches usually store volume bricks (actual voxel data) in a singe large 3D cache texture (or brick pool).

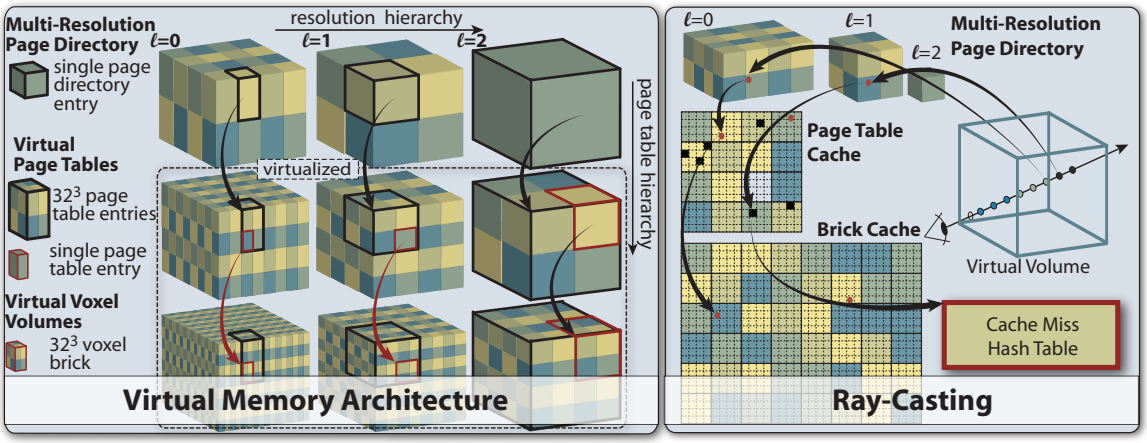

Figure 5: ***Multi-resolution, multi-level GPU page tables*** *[HBJP12]. The virtual memory architecture comprises two orthogonal hierarchies: the resolution hierarchy, and the page table hierarchy. Ray-casting performs address translation based on the multi-resolution page directory (i.e., one page directory per volume resolution) and shared "mixed-resolution" cache textures.*

If ray traversal needs to follow tree nodes (as in octree-based renderers), the working set of tree nodes must also be stored, e.g., in a *node pool* (e.g., [CNLE09, Eng11]).

If ray traversal is built on virtual to physical address translation (as in page table-based renderers), the working set of page table entries must be stored, e.g., in a *page table cache* (e.g., [BHL*11, HBJP12]).

4.3.1. Texture Cache Management

Texture allocation. Early tree-based volume renderers often employed one texture per brick, rendering one after the other in visibility order using one rendering pass per brick/tree node [LHJ99, WWH*00, GGSe*02, GS04]. However, multi-pass approaches are usually less performant than single-pass approaches and are also limited in the number of passes they can efficiently perform. To circumvent rendering bottlenecks due to many rendering passes, Hong et al. [HFK05] cluster bricks in layers (based on the manhattan distance) and render all bricks of the same layer at the same time.

To support single-pass rendering, bricking approaches and modern ray-guided renderers usually use a single large 3D cache texture (or brick pool) to store the working set [BHWB07, CN09, HBJP12], and often assume that the working set will fit into GPU memory.

When the working set does not fit into GPU memory, either the level of detail and thus the number of bricks in the working set can be reduced [HBJP12], or the renderer can switch to a multi-pass fall-back [Eng11, FSK13].

Texture updates. Whenever the working set changes, the cache textures have to be updated accordingly. Hadwiger et al. [HBJP12] compare texture update complexity between octree-based and multi-level page table approaches. Octree-based approaches usually have to do a large number of updates of small texture elements, whereas hierarchical page tables tend to perform fewer but larger updates.

To avoid cache thrashing [HP11], different brick replacement strategies have been introduced. Most common is the LRU scheme which replaces the brick in the cache that was least recently used [GMG08, CN09, FSK13]. It is also common to use a hybrid LRU/MRU scheme, where the LRU scheme is used unless the cache is too small for the current working set. In the latter case, the scheme is switched to MRU (most recently used) to reduce cache thrashing.

4.3.2. Virtual Texturing and Address Translation

Page tables. Kraus and Ertl [KE02] were the first to introduce adaptive texture maps for GPUs, where an image or volume can be stored in a bricked fashion with adaptive resolution and accessed via a look-up in a small index texture. This index texture can be seen as a page table [HP11].

Virtual texturing. Going further in this direction leads to virtual texturing [OVS12], also called Megatextures in game engines [vW09], and partially resident textures [BSH12]. A single, very large virtual texture is used for all data instead of allocating many small textures.

During rendering, virtual texture coordinates have to be translated to physical texture coordinates. Recently, hardware implementations of this scheme have become available with the OpenGL `GL_ARB_sparse_texture` extension. Unfortunately, current hardware limitations still limit the size of these textures to 16k pixels/voxels and do not allow for automatic page fault handling.

GPU-based page tables for virtual texturing are conceptually very similar to CPU virtual memory architectures [HP11]. For volume rendering, the virtual volume is decomposed into smaller bricks (i.e, pages), and a look-up

texture (i.e., page table) maps from virtual pages to physical pages. The earliest uses of virtual texturing and page tables in volume rendering [HSSB05] used a single page table texture. However, the basic concept of virtualization can be extended in a "recursive" fashion, which leads to a *page table hierarchy*. Virtual texturing architectures using such multi-level page tables have been shown to scale to volume data of extreme scale [BHL*11, HBJP12].

Hadwiger et al. [HBJP12] describe multi-level, multi-resolution page tables as a (conceptually) orthogonal 2D structure (see Figure 5, left). One dimension corresponds to the page table hierarchy, consisting of the page directory (the top-level page table) and several page tables below. The second dimension corresponds to the different resolution levels of the data. Each resolution level conceptually has its own page table hierarchy. However, the actual cache textures can be shared between all resolution levels. Multi-level page tables scale very well. For example, two levels have been shown to support volumes of up to several hundred terabytes, and three levels should in principle be sufficient even for exascale data [HBJP12] (in terms of "addressability").

Octrees. To traverse an octree directly on the GPU, not only the volume brick data, but also a (partial) tree needs to be stored on the GPU. Gobbetti et al. [GMG08] use a spatial index structure to store the current subtree with neighbor information. Each octree node stores pointers to its eight children and its six neighbors (via ropes [HBZ98]), and a pointer to the volume brick data. Crassin et al. [CN09, CNLE09] use an N^3 tree, whose current subtree is stored in a node pool and a brick pool, respectively. Each node stored in the node pool contains one pointer to its N^3 children, and one pointer to the corresponding volume brick in the brick pool (see Figure 4). Using a single child pointer is possible because the children are stored together in the node pool.

Hash tables. An alternative data structure to GPU page tables are hash tables, which have not yet received a lot of attention for large-scale volume rendering. However, Hastings et al. [HMG05] use spatial hashing to optimize collision detection in real-time simulations, and Nießner et al. [NZIS13] use voxel hashing for real-time 3D reconstruction.

4.4. Rendering (Ray Traversal)

In this section we will look into details of the actual rendering methods and how dynamic address translation is performed on the GPU.

Single-pass vs. multi-pass. In single-pass approaches the volume is traversed in front-to-back order in a single rendering pass as compared to multi-pass approaches that require multiple rendering passes. As mentioned before, the first GPU volume rendering approaches [CN93, CCF94, WE98, RSEB*00, HBH03], including the first octree-based renderers [LHJ99, WWH*00, GGSe*02, GS04, HFK05], were all based on multi-pass rendering. With the introduction of dynamic branching and looping on GPUs, single-pass approaches have been introduced to volume ray-casting [HSSB05, SSKE05].

Multi-pass approaches offer a higher flexibility, however, they also have a significant management overhead compared to single-pass rendering (i.e., context switching, final compositing) and usually result in lower performance. Furthermore, optimization techniques like early ray termination are not trivial in multi-pass rendering and create an additional overhead. Therefore, most state-of-the art ray-guided volume renderers use single-pass rendering [CNLE09, Eng11, HBJP12]. A limitation of single-pass approaches, however, is the requirement for the entire working set to fit into the cache. One way to circumvent this requirement is to use single-pass rendering as long as the working sets fits into the cache, and to switch to multi-pass rendering when the working set gets too large [Eng11, FSK13].

Multi-resolution rendering. There are several motivations for multi-resolution rendering. Next to the obvious advantage of data reduction and rendering speed-ups, choosing a resolution that matches the current screen resolution reduces aliasing artifacts due to undersampling [Wil83].

A multi-resolution data structure requires level-of-detail (LOD) or scale selection [LB03] for rendering. Weiler et al. [WWH*00] us a focus point oracle based on the distance from the center of a brick to a user-defined focus point to select a brick's LOD. Other methods to select a level of detail include estimating the screen-space error [GS04], using a combined factor of data homogeneity and importance [BNS01] or using the predicted visual significance of a brick [Lju06b]. A common method estimates the projected screen space size of the corresponding voxel/brick [CNLE09]. Whereas LOD selection is often performed on a per-brick basis, Hadwiger et al. [HBJP12] select the LOD on a per-sample basis for finer LOD granularity (see Figure 3).

The most common data refinement strategy (e.g., when quickly zooming-in on the data) consists of a "greedy" approach that iteratively loads the next higher-resolution of the brick until the desired resolution is reached [CNLE09]. A different approach, where the highest resolution is loaded directly and intermediate resolutions are skipped was proposed in [HBJP12]. Most recently, Fogal et al. [FSK13] found that the "greedy" approach converges in the fewest number of frames in their ray-guided ray-caster.

4.4.1. Virtual Texturing and Address Translation

Address translation is performed during ray-casting, when stepping along a ray, to access the correct location of a sample along the ray in the texture cache. When using multi-resolution data this implies that a GPU multi-resolution data structure has to be traversed dynamically on the GPU.

Tree traversal. Traversal algorithms for efficiently navigating and traversing trees, such as kd-trees or octrees have been well researched in the ray-tracing community. Amanatides and Woo [AW87] were the first to introduce a fast regular grid traversal algorithm. Recently, stackless traversal methods such as kd-restart [FS05] have received a lot of attention [HSHH07, PGS*07, HL09], as they are well-suited for GPU implementation.

The GPU octree traversal in Gobbetti et al. [GMG08] is based on previous work on rope trees [HBZ98, PGS*07], whereas Gigavoxels [CNLE09, CNSE10] and similar systems [Eng11, RTW13] base their octree traversal on the kd-restart algorithm [FS05].

Page table look-ups. In virtual texturing approaches [vW09, OVS12, HBJP12], each texture sample requires address translation from a virtual texture coordinate to a corresponding physical texture coordinate during rendering. This translation is done via small look-up texture(s), the *page table(s)*.

In multi-level page tables, additional levels of page tables are added [BHL*11]. The top level is usually called the *page directory*, in analogy to CPU virtual memory [HP11]. The right part of Figure 5 depicts address translation during ray-casting with a multi-resolution, multi-level page table. Hadwiger et al. [HBJP12] use this approach for rendering extreme-scale electron microscopy data. Their approach starts with computing a LOD for the current sample, which is then used to look up the page directory corresponding to that resolution. Next, address translation traverses the page table hierarchy from the page directory through the page table levels below. Previous page directory and page table look-ups can be cached to exploit spatial coherence. Thus, the number of texture look-ups that is required in practice is very low.

Handling missing and empty bricks. In contrast to traditional ray-casting approaches, where the working set is computed prior to rendering on the CPU, ray-guided volume renders only build up the current working set during ray traversal. This implies that ray-guided volume renderers have to be able to deal with missing bricks in GPU memory, because bricks are only requested and downloaded once they have been hit during ray-casting.

Whenever the ray-caster detects a missing brick (i.e., either a page table entry that is flagged as *unmapped* or a missing octree node), a request for that missing brick is written out. Crassin et al. [CN09] use multiple render targets to report missing nodes and then stop ray traversal. More recent approaches [CNSE10, HBJP12, FSK13] use OpenGL extensions such as `GL_ARB_shader_image_load_store` or CUDA, and often GPU hash tables, to report cache misses. Missing bricks can be either skipped, or substituted by a brick of lower resolution. After missing bricks are detected and reported, the CPU takes care of loading the missing data, downloading it into GPU memory, and updating the corresponding GPU data structures.

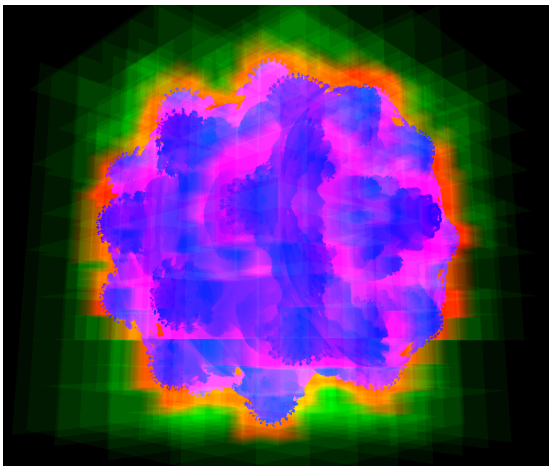

Figure 6: *Ray-guided volume rendering [FSK13] of the Mandelbulb data set. Colors indicate the amount of empty space skipping and sampling that needs to be performed (green: skipped empty brick, red: densely sampled brick, blue: densely sampled but quickly saturated). Image courtesy of Tom Fogal.*

Empty space skipping. In addition to skipping missing bricks, a common optimization strategy that is easily implemented in ray-guided volume rendering is empty space skipping. This optimization relies on knowing which bricks are empty bricks (e.g., by a flag in the page table) and skipped during ray-casting. Figure 6 shows a rendering with color-coded empty space skipping information.

5. Discussion and Conclusions

In this survey we have discussed different large-scale GPU-based volume rendering methods with an emphasis on ray-guided approaches. Over recent years, sophisticated scalable GPU volume visualization methods have been developed, hand in hand with the increased versatility and programmability of graphics hardware. GPUs nowadays support dynamic branching and looping, efficient read-back mechanisms to transfer data back from the GPU to the CPU, and several high-level APIs like CUDA or OpenCL to make GPU programming more efficient and enjoyable.

Our discussion of scalability in volume rendering was based on the notion of working sets. We assume that the data will never fit into GPU memory in its entirety. Therefore, it is crucial to determine, store, and render the working set of visible bricks in the current view efficiently and accurately. The review of "traditional" GPU volume rendering methods showed that these approaches have several shortcomings that severely limit their scalability. Traditionally, the working set of active bricks is determined on the CPU and no read-back mechanism is used to refine this working set. Additionally, due to previously limited branching or looping functionality

on GPUs, renderers often had to resort to multi-pass rendering approaches. Modern ray-guided approaches exhibit better scalability, they support dynamic traversal of multi-resolution structures on the GPU, and they allow determining the working set of active bricks based on actual visibility by employing efficient read-back mechanisms from the GPU to the CPU. Therefore, ray-guided approaches are promising for the future, where data set sizes will continue to increase.

In this survey we have focused on GPU-based approaches for single stand-alone workstations. However, there is a huge area of parallel and distributed visualization research that focuses on clusters, in-situ setups and client/server systems. Additionally, we expect web-based visualization to become more and more important, which will make it necessary to research scalable algorithms for remote visualization and mobile devices. Finally, as data sets get larger and scalable volume rendering methods more mature, it will become more and more important to have efficient workflows and integrated solutions that encompass the whole data flow through a system, from data acquisition and pre-processing to interactive visualization and analysis.

6. Acknowledgments

We would like to thank Fabio Marton, Timo Ropinski, and Daniel Weiskopf for their valuable feedback, and Hendrik Strobelt for his help. This work was partially supported by NSF grant OIA 1125087.

References

[AAM*11] AHERN S., ARIE S., MA K.-L., CHOUDHARY A., CRITCHLOW T., KLASKY S., PASCUCCI V., AHRENS J., BETHEL W. E., CHILDS H., HUANG J., JOY K., KOZIOL Q., LOFSTEAD G., MEREDITH J. S., MORELAND K., OSTROUCHOV G., PAPKA M., VISHWANATH V., WOLF M., WRIGHT N., WU K.: *Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization*. Tech. rep., Department of Energy, 2011. 1, 4, 7

[ALN*08] AHRENS J., LO L.-T. L.-T., NOUANESENGSY B., PATCHETT J., MCPHERSON A.: Petascale Visualization: Approaches and Initial Results. In *Workshop on Ultrascale Visualization, 2008. UltraVis '08.* (2008), pp. 24–28. 4

[AM00] ASSARSSON U., MOLLER T.: Optimized View Frustum Culling Algorithms for Bounding Boxes. *Journal of Graphics Tools 5*, 1 (Jan. 2000), 9–22. 11

[AMHH08] AKENINE-MÖLLER T., HAINES E., HOFFMAN N.: *Real-Time Rendering 3rd Edition*. A. K. Peters; Ltd., 2008. 11

[AW87] AMANATIDES J., WOO A.: A Fast Voxel Traversal Algorithm for Ray Tracing. In *Eurographics '87* (1987), pp. 3–10. 5, 14

[AWS92] AHLBERG C., WILLIAMSON C., SHNEIDERMAN B.: Dynamic Queries for Information Exploration: an Implementation and Evaluation. In *SIGCHI Conference on Human Factors in Computing Systems* (1992), CHI '92, pp. 619–626. 7

[BAaK*13] BEYER J., AL-AWAMI A., KASTHURI N., LICHTMAN J. W., PFISTER H., HADWIGER M.: ConnectomeExplorer: Query-Guided Visual Analysis of Large Volumetric Neuroscience Data. *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE SciVis '13) 19*, 12 (2013), 2868–2877. 7, 8

[BCH12] BETHEL E. W., CHILDS H., HANSEN C.: High Performance Visualization – Enabling Extreme-Scale Scientific Insight. Chapman & Hall, CRC Computational Science. CRC Press/Francis–Taylor Group, Nov. 2012. 1, 2, 3, 6, 7

[BG05] BRUCKNER S., GRÖLLER M.: Volumeshop: An Interactive System for Direct Volume Illustration. In *IEEE Visualization '05* (2005), pp. 671–678. 8

[BHAA*13] BEYER J., HADWIGER M., AL-AWAMI A., JEONG W.-K., KASTHURI N., LICHTMAN J., PFISTER H.: Exploring the Connectome - Petascale Volume Visualization of Microscopy Data Streams. *IEEE Computer Graphics and Applications 33*, 4 (2013), 50–61. 2, 4, 5, 7

[BHL*11] BEYER J., HADWIGER M., LICHTMAN J., REID R. C., JEONG W.-K., PFISTER H.: Demand-Driven Volume Rendering of Terascale EM Data. In *SIGGRAPH '11: Technical talk* (2011). 5, 12, 13, 14

[BHMF08] BEYER J., HADWIGER M., MÖLLER T., FRITZ L.: Smooth Mixed-Resolution GPU Volume Rendering. In *IEEE International Symposium on Volume and Point-Based Graphics (VG '08)* (2008), pp. 163–170. 5, 8, 10

[BHWB07] BEYER J., HADWIGER M., WOLFSBERGER S., BÜHLER K.: High-Quality Multimodal Volume Rendering for Preoperative Planning of Neurosurgical Interventions. *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE Visualization '07)* (2007), 1696–1703. 4, 8, 9, 11, 12

[BLK*11] BOCK D., LEE W.-C., KERLIN A., ANDERMANN M., HOOD G., WETZEL A., YURGENSON S., SOUCY E., KIM H. S., REID R. C.: Network Anatomy and In Vivo Physiology of Visual Cortical Neurons. *Nature 471*, 7337 (2011), 177–182. 1

[BNS01] BOADA I., NAVAZO I., SCOPIGNO R.: Multiresolution Volume Visualization with a Texture-based Octree. *The Visual Computer 17*, 3 (2001), 185–197. 5, 13

[BSH12] BILODEAU B., SELLERS G., HILLESLAND K.: AMD GPU Technical Publications: Partially Resident Textures (PRT) in the Graphics Core Next, 2012. 12

[BSS00] BARTZ D., SCHNEIDER B.-O., SILVA C.: Rendering and Visualization in Parallel Environments. *SIGGRAPH '00 course notes* (2000). 2, 6

[BvG*09] BRUCKNER S., ŠOLTÉSZOVÁ V., GRÖLLER M. E., HLADUVKA J., BÜHLER K., YU J., DICKSON B.: BrainGazer - Visual Queries for Neurobiology Research. *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE Visualization '09) 15*, 6 (Nov. 2009), 1497–1504. 7

[CBB*05] CHILDS H., BRUGGER E., BONNELL K., MEREDITH J., MILLER M., WHITLOCK B., MAX N.: A Contract-Based System For Large Data Visualization. In *IEEE Visualization '05* (2005), pp. 190–198. 9

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *IEEE Symposium on Volume Visualization* (1994), pp. 91–98. 8, 9, 13

[CKS03] CORREA W., KLOSOWSKI J. T., SILVA C.: Visibility-Based Prefetching for Interactive Out-Of-Core Rendering. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003), pp. 1–8. 6

[CMC*06] CASTANIE L., MION C., CAVIN X., LEVY B., BRUNO L., CASTANI L.: Distributed Shared Memory for Roaming Large Volumes. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1299–1306. 9

[CN93] CULLIP T., NEUMANN U.: Accelerating Volume Reconstruction with 3D Texture Hardware. In *Technical Report TR93-027, University of North Carolina at Chapel Hill* (1993). 8, 9, 13

[CN09] CRASSIN C., NEYRET F.: Beyond Triangles : Gigavoxels Effects In Video Games. In *SIGGRAPH '09: Technical talk* (2009). 5, 10, 11, 12, 13, 14

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: GigaVoxels : Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2009), Lecture Notes in Computer Science, pp. 15–22. 2, 3, 5, 7, 8, 12, 13, 14

[CNSE10] CRASSIN C., NEYRET F., SAINZ M., EISEMANN E.: Efficient Rendering of Highly Detailed Volumetric Scenes with GigaVoxels. In *GPU Pro*. A. K. Peters; Ltd, 2010, ch. X.3, pp. 643–676. 10, 11, 14

[CPA*10] CHILDS H., PUGMIRE D., AHERN S., WHITLOCK B., HOWISON M., PRABHAT, WEBER G., BETHEL E.: Extreme Scaling of Production Visualization Software on Diverse Architectures. *IEEE Computer Graphics and Applications 30*, 3 (2010), 22–31. 3

[CSK*11] CONGOTE J., SEGURA A., KABONGO L., MORENO A., POSADA J., RUIZ O.: Interactive Visualization of Volumetric Data with WebGL in Real-Time. In *16th International Conference on 3D Web Technology - Web3D '11* (2011), pp. 137–146. 9

[DKR97] DERTHICK M., KOLOJEJCHICK J., ROTH S. F.: An Interactive Visual Query Environment for Exploring Data. In *Tenth Annual ACM Symposium on User Interface Software and Technology (UIST '97)* (1997), pp. 189–198. 7

[EHK*06] ENGEL K., HADWIGER M., KNISS J. M., REZK-SALAMA C., WEISKOPF D.: *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006. 2

[Eng11] ENGEL K.: CERA-TVR: A Framework for Interactive High-Quality Teravoxel Volume Visualization on Standard PCs. In *Large-Data Analysis and Visualization, (LDAV '11 Posters)* (2011). 2, 8, 9, 10, 12, 13, 14

[EPMS09] EILEMANN S., PAJAROLA R., MAKHINYA M., SOCIETY I. C.: Equalizer: A Scalable Parallel Rendering Framework. *IEEE Transactions on Visualization and Computer Graphics 15*, 3 (2009), 436–452. 9

[ESE00] ENGEL K., SOMMER O., ERTL T.: A Framework for Interactive Hardware Accelerated Remote 3D-Visualization. In *TCVG Symposium on Visualization (VisSym '00)* (2000), pp. 167–177. 9

[FCS*10] FOGAL T., CHILDS H., SHANKAR S., KRÜGER J., BERGERON R. D., HATCHER P.: Large Data Visualization on Distributed Memory Multi-GPU Clusters. In *High Performance Graphics* (2010), pp. 57–66. 5, 9

[FK05] FRANK S., KAUFMAN A.: Distributed Volume Rendering on a Visualization Cluster. In *Ninth International Conference on Computer Aided Design and Computer Graphics* (2005), pp. 5–10. 6

[FK10] FOGAL T., KRÜGER J.: Tuvok - An Architecture for Large Scale Volume Rendering. In *15th Vision, Modeling and Visualization Workshop '10* (2010), pp. 139–146. 4, 8

[FM12] FOUT N., MA K.-L.: An Adaptive Prediction-Based Approach to Lossless Compression of Floating-Point Volume Data. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (2012), 2295–2304.

[FS05] FOLEY T., SUGERMAN J.: KD-Tree Acceleration Structures for a GPU Raytracer. In *Graphics Hardware* (2005), pp. 15–22. 5, 10, 14

[FSK13] FOGAL T., SCHIEWE A., KRÜGER J.: An Analysis of Scalable GPU-Based Ray-Guided Volume Rendering. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV '13)* (2013), pp. 43–51. 2, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14

[FW08] FALK M., WEISKOPF D.: Output-Sensitive 3D Line Integral Convolution. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (2008), 820–834. 2

[GGSe*02] GUTHE S., GONSER J., STRASS ER W., WAND M., STRAER W.: Interactive Rendering of Large Volume Data Sets. In *IEEE Visualization* (2002), pp. 53–59. 5, 8, 9, 12, 13

[GHSK03] GAO J., HUANG J., SHEN H.-W., KOHL J. A.: Visibility Culling Using Plenoptic Opacity Functions for Large Volume Visualization. In *IEEE Visualization '03* (2003), pp. 341–348. 11

[GKM93] GREENE N., KASS M., MILLER G.: Hierarchical Z-Buffer Visibility. In *SIGGRAPH '93* (1993), pp. 231–238. 2, 11

[GM05] GOBBETTI E., MARTON F.: Far Voxels: A Multiresolution Framework for Interactive Rendering of Huge Complex 3D Models on Commodity Graphics Platforms. *ACM Transactions on Graphics 24*, 3 (2005), 878–885. 5

[GMG08] GOBBETTI E., MARTON F., GUITI I.: A Single-Pass GPU Ray Casting Framework for Interactive Out-of-Core Rendering of Massive Volumetric Datasets. *The Visual Computer 24*, 7 (2008), 787–806. 5, 8, 10, 11, 12, 13, 14

[GS04] GUTHE S., STRASSER W.: Advanced Techniques for High-Quality Multi-Resolution Volume Rendering. *Computers & Graphics 28*, 1 (2004), 51–58. 5, 8, 9, 12, 13

[GSHK04] GAO J., SHEN H.-W., HUANG J., KOHL J. A.: Visibility Culling for Time-Varying Volume Rendering Using Temporal Occlusion Coherence. In *IEEE Visualization '04* (2004), pp. 147–154. 11

[HBH03] HADWIGER M., BERGER C., HAUSER H.: High-Quality Two-Level Volume Rendering of Segmented Data Sets on Consumer Graphics Hardware. In *IEEE Visualization '03* (2003), pp. 301–308. 8, 9, 13

[HBJP12] HADWIGER M., BEYER J., JEONG W.-K., PFISTER H.: Interactive Volume Exploration of Petascale Microscopy Data Streams Using a Visualization-Driven Virtual Memory Approach. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE of SciVis '12) 18*, 12 (2012), 2285–2294. 2, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14

[HBZ98] HAVRAN V., BITTNER J., ZÁRA J.: Ray Tracing With Rope Trees. In *14th Spring Conference On Computer Graphics* (1998), pp. 130–139. 13, 14

[Hec86] HECKBERT P.: Survey of Texture Mapping. *IEEE Computer Graphics and Applications 6*, 11 (1986), 56–67. 9

[Hel13] HELMSTAEDTER M.: Cellular-Resolution Connectomics: Challenges of Dense Neural Circuit Reconstruction. *Nature Methods 10*, 6 (June 2013), 501–7. 1

[HFK05] HONG W., FENG Q., KAUFMAN A.: GPU-Based Object-Order Ray-Casting for Large Datasets. In *Eurographics/IEEE VGTC Workshop on Volume Graphics '05* (2005), pp. 177–240. 8, 9, 12, 13

[HL09] HUGHES D. M., LIM I. S.: Kd-Jump: A Path-Preserving Stackless Traversal for Faster Isosurface Raytracing on GPUs. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1555–1562. 5, 14

[HMG05] HASTINGS E. J., MESIT J., GUHA R. K.: Optimization of Large-Scale , Real-Time Simulations by Spatial Hashing. In *Summer Computer Simulation Conference* (2005), pp. 9–17. 13

[HN12] HEITZ E., NEYRET F.: Representing Appearance and Pre-Filtering Subpixel Data in Sparse Voxel Octrees. In *ACM SIGGRAPH / Eurographics conference on High-Performance Graphics (EGGH-HPG '12)* (2012), pp. 125–134. 5

[HP11] HENNESSEY J. L., PATTERSON D. A.: *Computer Architecture: A Quantitative Approach*, fifth ed. Morgan Kaufmann, 2011. 12, 14

[HSB*12] HADWIGER M., SICAT R., BEYER J., KRÜGER J., MÖLLER T.: Sparse PDF Maps for Non-Linear Multi-Resolution Image Operations. In *ACM Transactions on Graphics (Proc. of ACM SIGGRAPH Asia '12)* (2012), pp. 198:1–198:12. 2, 5

[HSHH07] HORN D. R., SUGERMAN J., HOUSTON M., HANRAHAN P.: Interactive k-d Tree GPU Raytracing. In *Symposium on Interactive 3D Graphics and Games - I3D '07* (2007), p. 167. 5, 14

[HSSB05] HADWIGER M., SIGG C., SCHARSACH H., BÜHLER K.: Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces. *Computer Graphics Forum (Proc. of Eurographics '05) 24*, 3 (2005), 303–312. 8, 9, 11, 13

[ILC10] ISENBURG M., LINDSTROM P., CHILDS H.: Parallel and Streaming Generation of Ghost Data for Structured Grids. *IEEE Computer Graphics & Applications 30*, 3 (2010), 32–44. 4, 6

[JBH*09] JEONG W.-K. W.-K., BEYER J., HADWIGER M., VASQUEZ A., PFISTER H., WHITAKER R. T., VAZQUEZ A.: Scalable and Interactive Segmentation and Visualization of Neural Processes in EM Datasets. *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE Visualization '09) 15*, 6 (2009), 1505–1514. 7, 8, 9

[JJY*11] JEONG W.-K., JOHNSON M. K., YU I., KAUTZ J., PFISTER H., PARIS S.: Display-Aware Image Editing. In *IEEE International Conference on Computational Photography (ICCP '11)* (Apr. 2011), IEEE, pp. 1–8. 2

[JST*10] JEONG W.-K., SCHNEIDER J., TURNEY S. G., FAULKNER-JONES B. E., MEYER D., WESTERMANN R., REID C., LICHTMAN J., PFISTER H.: Interactive Histology of Large-Scale Biomedical Image Stacks. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1386–1395. 2, 6

[KAL*11] KLASKY S., ABBASI H., LOGAN J., PARASHAR M., SCHWAN K., SHOSHANI A., WOLF M., SEAN A., ALTINTAS I., BETHEL W., LUIS C., CHANG C., CHEN J., CHILDS H., CUMMINGS J., DOCAN C., EISENHAUER G., ETHIER S., GROUT R., LAKSHMINARASIMHAN S., LIN Z., LIU Q., MA X., MORELAND K., PASCUCCI V., PODHORSZKI N., SAMATOVA N., SCHROEDER W., TCHOUA R., TIAN Y., VATSAVAI R., WU J., YU W., ZHENG F.: In Situ Data Processing for Extreme-Scale Computing. In *SciDAC Conference* (2011). 7

[KE02] KRAUS M., ERTL T.: Adaptive Texture Maps. In *Graphics Hardware* (2002), pp. 7–15. 9, 12

[KGB*09] KAINZ B., GRABNER M., BORNIK A., HAUSWIESNER S., MUEHL J., SCHMALSTIEG D.: Ray Casting of Multiple Volumetric Datasets with Polyhedral Boundaries on Manycore GPUs. *ACM Transactions on Graphics 28*, 5 (2009), 1–9. 8, 9

[KH13] KEHRER J., HAUSER H.: Visualization and Visual Analysis of Multifaceted Scientific Data: A Survey. *IEEE Transactions on Visualization and Computer Graphics 19*, 3 (Mar. 2013), 495–513. 2

[KMS*06] KASIK D., MANOCHA D., STEPHENS A., BRUDERLIN B., SLUSALLEK P., GOBBETTI E., CORREA W., QUILEZ I.: Real Time Interactive Massive Model Visualization. *Eurographics '06: Tutorials* (2006). 2, 6

[Kno06] KNOLL A.: A Survey of Octree Volume Rendering Methods. In *First IRTG workshop* (2006). 5

[KTW*11] KNOLL A., THELEN S., WALD I., HANSEN C. D., HAGEN H., PAPKA M. E.: Full-Resolution Interactive CPU Volume Rendering with Coherent BVH Traversal. In *IEEE Pacific Visualization Symposium '11* (Mar. 2011), pp. 3–10. 10

[KW03] KRÜGER J., WESTERMANN R.: Acceleration Techniques for GPU-based Volume Rendering. In *IEEE Visualization '03* (2003), pp. 287–292. 8, 9

[LB03] LINDEBERG T., BRETZNER L.: *Real-Time Scale Selection in Hybrid Multi-Scale Representations*. Tech. rep., KTH (Royal Institute of Technology), 2003. 13

[LCD09] LIU B., CLAPWORTHY G. J., DONG F.: Accelerating Volume Raycasting using Proxy Spheres. *Computer Graphics Forum (Proc. of EuroVis '09) 28*, 3 (June 2009), 839–846. 9

[LHJ99] LAMAR E., HAMANN B., JOY K. I.: Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In *IEEE Visualization '99* (1999), pp. 355–362. 2, 5, 8, 9, 12, 13

[Lju06a] LJUNG P.: Adaptive Sampling in Single Pass, GPU-based Raycasting of Multiresolution Volumes. In *Eurographics/IEEE VGTC Workshop on Volume Graphics '06* (2006), pp. 39–46. 4, 5, 8, 9

[Lju06b] LJUNG P.: *Efficient Methods for Direct Volume Rendering of Large Data Sets*. PhD thesis, Linköping University, Sweden, 2006. 5, 13

[LK10a] LAINE S., KARRAS T.: Efficient Sparse Voxel Octrees. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '10)* (2010), pp. 55–63. 5

[LK10b] LAINE S., KARRAS T.: *Efficient Sparse Voxel Octrees - Analysis , Extensions , and Implementation*. Tech. rep., NVIDIA, 2010. 5

[LKHW04] LEFOHN A. E., KNISS J. M., HANSEN C. D., WHITAKER R. T.: A Streaming Narrow-Band Algorithm: Interactive Computation and Visualization of Level Sets. *IEEE Transactions on Visualization and Computer Graphics 10*, 4 (2004), 422–433. 6

[LMK03] LI W., MUELLER K., KAUFMAN A.: Empty Space Skipping and Occlusion Clipping for Texture-based Volume Rendering. In *IEEE Visualization '03* (2003), pp. 317–324. 2, 8, 11

[MAWM11] MOLONEY B., AMENT M., WEISKOPF D., MÖLLER T.: Sort-First Parallel Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics 17*, 8 (2011), 1164–1177. 9

[MCE*94] MOLNAR S., COX M., ELLSWORTH D., FUCHS H., ANDN D. ELLSWORTH M. C.: A Sorting Classification of Parallel Rendering. *IEEE Computer Graphics & Applications 14*, 4 (1994), 23–32. 6

[MHE01] MAGALLÓN M., HOPF M., ERTL T.: Parallel Volume Rendering Using PC Graphics Hardware. In *Pacific Conference on Computer Graphics and Applications* (2001), pp. 384–389. 9

[MHS08] MARSALEK L., HAUBER A., SLUSALLEK P.: High-Speed Volume Ray Casting with CUDA. In *IEEE Symposium on Interactive Ray Tracing* (Aug. 2008), p. 185. 8, 9

[ML13] MORGAN J. L., LICHTMAN J. W.: Why Not Connectomics? *Nature Methods 10*, 6 (June 2013), 494–500. 1

[MM10] MARCHESIN S. S., MA K.-L.: Cross-Node Occlusion in Sort-Last Volume Rendering. In *Eurographics Symposium on Parallel Graphics and Visualization* (2010), pp. 11–18. 11

[MOM*11] MORELAND K., OLDFIELD R., MARION P., JOURDAIN S., PODHORSZKI N., VISHWANATH V., FABIAN N., DOCAN C., PARASHAR M., HERELD M., PAPKA M. E., KLASKY S.: Examples of In Transit Visualization. In *Second International Workshop on Petascale Data Analytics: Challenges and Opportunities (PDAC '11)* (2011), pp. 1–6. 7

[Mor66] MORTON G. M.: *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. Tech. rep., IBM Ltd., 1966. 6

[Mor12] MORELAND K.: Oh, $#*@! Exascale! The Effect of Emerging Architectures on Scientific Discovery. *2012 SC Companion: High Performance Computing, Networking Storage and Analysis* (2012), 224–231. 4, 7

[Mor13] MORELAND K.: A Survey of Visualization Pipelines. *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE SciVis '13) 19*, 3 (Mar. 2013), 367–78. 4

[MRH08] MENSMANN J., ROPINSKI T., HINRICHS K.: Accelerating Volume Raycasting using Occlusion Frustums. In *Fifth EG/IEEE Conference on Point-Based Graphics* (2008), pp. 147–154. 9

[MRH10] MENSMANN J., ROPINSKI T., HINRICHS K. H.: An Advanced Volume Raycasting Technique using GPU Stream Processing. In *International Conference on Computer Graphics Theory and Applications (GRAPP '10)* (Angers, 2010), INSTICC Press, pp. 190–198. 8, 9

[MSE06] MÜLLER C., STRENGERT M., ERTL T.: Optimized Volume Raycasting for Graphics-Hardware-based Cluster Systems. In *Eurographics Symposium on Parallel Graphics and Visualization* (2006), pp. 59–66. 9

[Mur93] MURAKI S.: Volume Data and Wavelet Transforms. *IEEE Computer Graphics and Applications 13*, 4 (1993), 50–56. 5

[Mus13] MUSETH K.: VDB: High-Resolution Sparse Volumes with Dynamic Topology. *ACM Transactions on Graphics 32*, 3 (2013), 27:1–27:22. 5

[MW95] MARTIN A. R., WARD M. O.: High Dimensional Brushing for Interactive Exploration of Multivariate Data. In *IEEE Visualization '95* (1995), pp. 271–278. 7

[MWMS07] MOLONEY B., WEISKOPF D., MÖLLER T., STRENGERT M.: Scalable Sort-First Parallel Direct Volume Rendering with Dynamic Load Balancing. In *Eurographics Symposium on Parallel Graphics and Visualization* (2007), pp. 45–52. 9

[MWY*09] MA K.-L., WANG C., YU H., MORELAND K., HUANG J., ROSS R.: Next-Generation Visualization Technologies: Enabling Discoveries at Extreme Scale. In *SciDAC Review* (2009), pp. 12–21. 1, 4, 7

[Neu94] NEUMANN U.: Communication Costs for Parallel Volume-Rendering Algorithms. *IEEE Computer Graphics & Applications 14*, 4 (July 1994), 49–58. 6

[NVI13] NVIDIA CORPORATION: CUDA C Programming Guide, 2013. http://www.nvidia.com/object/cuda_get.html. 8

[NZIS13] NIESSNER M., ZOLLHÖFER M., IZADI S., STAMMINGER M.: Real-Time 3D Reconstruction at Scale Using Voxel Hashing. *ACM Transactions on Graphics 32*, 6 (2013), 1–11. 13

[OLG*07] OWENS J. D., LUEBKE D., GOVINDARAJU N., HARRIS M., KRÜGER J., LEFOHN A. E., PURCELL T. J., KR J.: A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum 26*, 1 (2007), 80–113. 8

[OVS12] OBERT J., VAN WAVEREN J., SELLERS G.: Virtual Texturing in Software and Hardware. In *SIGGRAPH '12 Courses* (2012). 5, 12, 14

[PF02] PASCUCCI V., FRANK R. J.: Hierarchical Indexing for Out-of-Core Access to Multi-Resolution Data. In *Hierarchical and Geometrical Methods in Scientific Visualization*. 2002, pp. 225–241. 6

[PGS*07] POPOV S., GÜNTHER J., SEIDEL H.-P. H.-P., SLUSALLEK P., GÜNTHER J.: Stackless Kd-Tree Traversal for High Performance GPU Ray Tracing. *Eurographics 26*, 3 (2007), 415–424. 5, 14

[PHKH04] PROHASKA S., HUTANU A., KAHLER R., HEGE H.-C.: Interactive Exploration of Large Remote Micro-CT Scans. In *IEEE Visualization* (2004), pp. 345–352. 8, 9, 10

[PJ95] PARKER S. G., JOHNSON C. R.: SCIRun : A Scientific Programming Environment for Computational Steering. In *ACM/IEEE conference on Supercomputing '95* (1995). 7

[PSL*98] PARKER S., SHIRLEY P., LIVNAT Y., HANSEN C., SLOAN P.: Interactive Ray Tracing for Isosurface Rendering. In *IEEE Visualization '98* (1998), pp. 233–238. 11

[RÖ9] RÖMISCH K.: *Sparse Voxel Octree Ray Tracing on the GPU*. PhD thesis, Aarhus University, 2009. 5

[RGG*13] RODRÍGUEZ M., GOBBETTI E., GUITAN J. A. I., MAKHINYA M., MARTON F., PAJAROLA R., SUTER S.: A Survey of Compressed GPU-Based Direct Volume Rendering. *Eurographics State of The Art Report (STAR)* (2013), 117–136. 2, 6

[RGW*03] ROETTGER S., GUTHE S., WEISKOPF D., ERTL T., STRASSER W.: Smart Hardware-Accelerated Volume Rendering. In *Symposium on Visualization (VISSYM '03)* (2003), pp. 231–238. 8, 9

[Ros06] ROST R. J.: *OpenGL Shading Language (2nd Edition)*. Addison-Wesley Professional, 2006. 8

[RSEB*00] REZK-SALAMA C., ENGEL K., BAUER M., GREINER G., ERTL T.: Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2000), pp. 109–118. 8, 9, 13

[RTW13] REICHL F., TREIB M., WESTERMANN R.: Visualization of Big SPH Simulations via Compressed Octree Grids. In *IEEE Big Data* (2013), pp. 71–78. 5, 8, 10, 14

[RV06] RUIJTERS D., VILANOVA A.: Optimizing GPU Volume Rendering. In *Winter School of Computer Graphics (WSCG '06)* (2006), pp. 9–16. 8

[SBH*08] SAMATOVA N. F., BREIMYER P., HENDRIX W., SCHMIDT M. C., RHYNE T.-M.: An Outlook Into Ultra-Scale Visualization of Large-Scale Biological Data. In *Workshop on Ultrascale Visualization, UltraVis 2008.* (2008), pp. 29–39. 4

[SCC*02] SILVA C., CHIANG Y.-J., CORREA W., EL-SANA J., LINDSTROM P.: Out-of-Core Algorithms for Scientific Visualization and Computer Graphics. In *IEEE Visualization '02 Course Notes* (2002). 6, 7

[Shn94] SHNEIDERMAN B.: Dynamic Queries for Visual Information Seeking. *IEEE Software 11*, 6 (1994), 70–77. 7

[SHN*06] SCHARSACH H., HADWIGER M., NEUBAUER A., WOLFSBERGER S., BÜHLER K.: Perspective Isosurface and Direct Volume Rendering for Virtual Endoscopy Applications. In *Eurovis/IEEE-VGTC Symposium on Visualization* (2006), pp. 315–323. 9, 11

[SO92] SHARIR M., OVERMARS M. H.: A Simple Output-sensitive Algorithm for Hidden Surface Removal. *ACM Trans. Graph. 11*, 1 (1992), 1–11. 2, 3

[SSJ*11] SUMMA B., SCORZELLI G., JIANG M., BREMER P.-T., PASCUCCI V.: Interactive Editing of Massive Imagery Made Simple. *ACM Transactions on Graphics 30*, 2 (Apr. 2011), 1–13. 6

[SSKE05] STEGMAIER S., STRENGERT M., KLEIN T., ERTL T.: A Simple and Flexible Volume Rendering Framework

for Graphics-Hardware-based Raycasting. *Eurographics/IEEE VGTC Workshop on Volume Graphics '05* (2005), 187–195. 8, 9, 13

[SSWB05] STOCKINGER K., SHALF J., WU K., BETHEL E. W.: Query-Driven Visualization of Large Data Sets. In *IEEE Visualization '05* (2005), pp. 167–174. 7

[THM01] TURLINGTON J. Z., HIGGINS W. E., MEMBER S.: New Techniques for Efficient Sliding Thin-Slab Volume Visualization. *IEEE Transactions on Medical Imaging 20*, 8 (2001), 823–835. 10

[TMJ98] TANNER C. C., MIGDAL C. J., JONES M. T.: The Clipmap : A Virtual Mipmap. In *SIGGRAPH '98* (1998), ACM, pp. 151–158. 5

[TTRU*06] TU T., TABORDA-RIOS R., URBANIC J., YU H., BIELAK J., GHATTAS O., LOPEZ J. C., MA K.-L., O'HALLARON D. R., RAMIREZ-GUZMAN L., STONE N.: Analytics Challenge - Remote Runtime Steering of Integrated Terascale Simulation and Visualization. In *ACM/IEEE conference on Supercomputing (SC '06)* (2006), ACM Press, p. 297. 7

[TYC*11] TIKHONOVA A., YU H., CORREA C. D., CHEN J. H., MA K.-L.: A Preview and Exploratory Technique for Large-Scale Scientific Simulations. In *Eurographics Conference on Parallel Graphics and Visualization (EGPGV'11)* (2011), pp. 111–120. 7

[VOS*10] VO H. T., OSMARI D. K., SUMMA B., COMBA J. A. L. D., PASCUCCI V., SILVA C. T.: Streaming-Enabled Parallel Dataflow Architecture for Multicore Systems. *Computer Graphics Forum 29*, 3 (2010), 1073–1082. 7

[vW09] VAN WAVEREN J. M. P.: id Tech 5 Challenges: From Texture Virtualization to Massive Parallelization. Talk in Beyond Programmable Shading course, SIGGRAPH '09, 2009. 5, 12, 14

[WE98] WESTERMANN R., ERTL T.: Efficiently Using Graphics Hardware in Volume Rendering Applications. In *SIGGRAPH '98* (1998), pp. 169–178. 8, 9, 13

[WGL*05] WANG C., GAO J., LI L., SHEN W.-W., SHEN H.-W.: A Multiresolution Volume Rendering Framework for Large-Scale Time-Varying Data Visualization. In *Eurographics/IEEE VGTC Workshop on Volume Graphics '05* (2005), pp. 11–223. 10

[Wil83] WILLIAMS L.: Pyramidal Parametrics. *Computer Graphics (Proc. of SIGGRAPH '83) 17*, 3 (1983), 1–11. 5, 13

[Wit98] WITTENBRINK C. M.: *Survey of Parallel Volume Rendering Algorithms*. Tech. rep., Hewlett-Packard Laboratories, 1998. 2, 6

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMAN K., ERTL T.: Level-Of-Detail Volume Rendering via 3D Textures. In *IEEE Symposium on Volume Visualization* (2000), pp. 7–13. 2, 3, 5, 8, 9, 12, 13

[YMC06] YOUNESY H., MÖLLER T., CARR H.: Improving the Quality of Multi-Resolution Volume Rendering. In *Eurovis/IEEE-VGTC Symposium on Visualization '06* (2006), pp. 251–258. 5

[YWG*10] YU H., WANG C., GROUT R. W., CHEN J. H., MA K.-L.: In Situ Visualization for Large-Scale Combustion Simulations. *IEEE Computer Graphics & Applications 30*, 3 (2010), 45–57. 7

[ZMHH97] ZHANG H., MANOCHA D., HUDSON T., HOFF K. E.: Visibility Culling Using Hierarchical Occlusion Maps. In *ACM SIGGRAPH '97* (1997), pp. 77–88. 2, 11

[ZSJ*05] ZHANG J., SUN J., JIN Z., ZHANG Y., ZHAI W., ZHAI Q.: Survey of Parallel and Distributed Volume Rendering: Revisited. In *International Conference on Computational Science and Its Applications (ICCSA '05)* (2005), vol. 3, pp. 435–444. 2, 6

Biography

Johanna Beyer is a postdoctoral fellow in the School of Engineering and Applied Sciences at Harvard University. Before joining Harvard, she was a postdoctoral fellow at the Visual Computing Center at King Abdullah University of Science and Technology. Her research interests include large-data visualization, parallel visualization, and GPU-based volume rendering for neuroscience and neurobiology. She received a PhD in computer science from the Vienna University of Technology in 2010.

Markus Hadwiger is an Assistant Professor in computer science at King Abdullah University of Science and Technology and head of the High-Performance Visualization group at the Visual Computing Center at KAUST. Before joining KAUST, he was a Senior Researcher at the VRVis Research Center in Vienna. His PhD is in computer science from the Vienna University of Technology in 2004. He is a co-author of the book Real-Time Volume Graphics. His research interests include petascale visual computing and scientific visualization, volume rendering, large-scale image processing, and GPU algorithms and architecture.

Hanspeter Pfister is An Wang Professor of Computer Science and Director of the Institute for Applied Computational Science at the Harvard School of Engineering and Applied Sciences. His research in visual computing lies at the intersection of visualization, computer graphics, and computer vision. It spans a wide range of topics, including biomedical visualization, image and video analysis, 3D fabrication, and data science. Dr. Pfister has a Ph.D. in Computer Science from the State University of New York at Stony Brook and an M.S. in Electrical Engineering from ETH Zurich, Switzerland. Before joining Harvard he worked for over a decade at Mitsubishi Electric Research Laboratories where he was Associate Director and Senior Research Scientist.