On Evaluating Runtime Performance of Interactive Visualizations

Valentin Bruder, Christoph Müller, Steffen Frey, and Thomas Ertl, *Fellow, IEEE*

Abstract—As our field matures, evaluation of visualization techniques has extended from reporting runtime performance to studying user behavior. Consequently, many methodologies and best practices for user studies have evolved. While maintaining interactivity continues to be crucial for the exploration of large data sets, no similar methodological foundation for evaluating runtime performance has been developed. Our analysis of 50 recent visualization papers on new or improved techniques for rendering volumes or particles indicates that only a very limited set of parameters like different data sets, camera paths, viewport sizes, and GPUs are investigated, which make comparison with other techniques or generalization to other parameter ranges at least questionable. To derive a deeper understanding of qualitative runtime behavior and quantitative parameter dependencies, we developed a framework for the most exhaustive performance evaluation of volume and particle visualization techniques that we are aware of, including millions of measurements on ten different GPUs. This paper reports on our insights from statistical analysis of this data, discussing independent and linear parameter behavior and non-obvious effects. We give recommendations for best practices when evaluating runtime performance of scientific visualization applications, which can serve as a starting point for more elaborate models of performance quantification.

Index Terms—Performance evaluation, scientific visualization, volume rendering, particle rendering

1 Introduction and Motivation

The analysis of algorithmic and computational performance is probably as old as research in computer science. In the field of scientific visualization, run-time performance has always been of great importance due to the high computational demand of many algorithms used in this domain on the one hand and the strive for interactive frame rates on the other. The traditional approach to performance evaluation in scientific visualization papers concerned with interactive techniques often is measuring performance for several data sets the authors deem representative. In many cases, authors compare the proposed technique with some comparable ones on one or two different systems (cf. Tables 2 & 4). The fact that there are numerous factors that can influence performance even for simple visualization algorithms raises the question to which extent such common evaluations are actually representative. Does using only a small subset of possible configurations such as a few data sets and one computing device lead to missing influential performance characteristics? Which are the most important factors that need to be investigated closely to paint a comprehensive picture of rendering performance? Are there typical correlations of factors that might help finding a concise, yet complete description?

To address these questions, we developed a benchmarking framework capable of running and measuring algorithms with different parameter configurations in an automated fashion (Section 4). In this work, we focus on GPU-accelerated interactive scientific visualization approaches running on a single machine. We discuss two case studies, investigating parameter dimensions such as data sets, GPUs, camera configurations, rendering techniques, and sampling rates.

The first case study concerns itself with volume rendering, a fundamental tool for the analysis of simulations and measurements in numerous scientific domains such as physics, medicine, engineering, etc. Volume rendering has been an active area of research in scientific visualization for decades, and many ways to improve its run-time performance have been proposed. From a technical point of view, it has very distinct properties, most importantly its reliance on fast graphics memory access and interpolation. The volume rendering case study is described in Section 5.

In the second case study, we consider GPU-based splatting of spherical glyphs, which has found wide adoption in the visualization of particle-based data sets like molecular dynamics simulations. Improving rendering performance has been a major focus of research in this area, because the ever-increasing size of simulations demands faster rendering to enable interactive data exploration. In context of our case study, the interesting aspects of this application are the (usually) incoherent layout of the data in memory, its reliance on the rasterization pipeline and it being rather compute-bound than memory-bound. The particle rendering case study is described in Section 6

Thus, our case studies cover algorithms, which are memory-bound and interpolation-bound on the one hand, as well as rasterization-bound on the other. For each case study, we provide a brief overview of performance evaluation in previous papers (Sections 5.1 and 6.1) and an outline of our implementation used in the benchmark (Sections 5.2

• *The authors are with the Visualization Research Center, University of Stuttgart, Stuttgart 70174, Germany. E-mail: {valentin.bruder, christoph.mueller, steffen.frey}@visus.uni-stuttgart.de, Thomas.Ertl@vis.uni-stuttgart.de.*

Manuscript received 14 Sept. 2018; revised 14 Dec. 2018; accepted 13 Jan. 2019. Date of publication 8 Feb. 2019; date of current version 4 Aug. 2020.
(Corresponding author: Valentin Bruder.)
Recommended for acceptance by I. Fujishiro.
Digital Object Identifier no. 10.1109/TVCG.2019.2898435

and 6.2). Afterwards, we present a top-down analysis of the measurements we obtained on a variety of discrete GPUs from NVIDIA and AMD (Sections 5.3 and 6.3). Thereby, a focus lies on identifying linear factors that can be considered independently from other variable parameters. Furthermore, finding factors with similar characteristics across case studies is another priority of our data analysis. On this basis, we provide a list of best practices (Section 7.1) for improving performance evaluations in future works. The main contributions of this work are:

- Development and provision of a publicly available benchmarking framework for scientific visualization applications, including plugins for volume rendering and particle rendering.
- Based on a review of state-of-the-art and extensive measurements with our framework, we suggest best practices that can be used as a guideline for future empirical performance evaluation of visualization techniques.

2 Related Work

Laramee [1] discusses some fundamentals of performance analysis in his educational paper on "How to write a visualization research paper". He advises to present a table with performance results including optimizations for speed and quality. Furthermore, he suggests to show the limits of the algorithm. This form of performance evaluation can be found in many visualization papers.

There are various papers encouraging the use of an empirical approach to performance evaluation, as this is the foundation for building prediction models. Tichy et al. [2] performed a quantitative study on experimental evaluation in computer science with the result that relatively few published papers contain experimentally validated results. Feitelson [3] argues that research projects tend to create their "own unique world". He proposes a higher acceptance of experimentation approaches from other scientific fields.

Interactivity in visualization is crucial to enable users to explore and gain new insights from data. To achieve the necessary low response times, GPUs with their high compute and memory performance are commonly employed for interactive scientific visualization [4], [5], [6]. For instance, enhancing the performance of GPU-based volumetric raycasting in different forms has emerged into a significant field of research on its own [7]. However, even in this well-studied context, the performance achieved in practice significantly varies with factors like data, parameters, and hardware, in a way that is almost impossible to assess beforehand. To account for this, Bethel et al. [8] tested a variety of settings, algorithmic optimizations, and different memory layouts for the Intel Nehalem, AMD Magny Cours, and NVIDIA Fermi architecture. Among other aspects, they found that optimal configurations vary across platforms, often in non-obvious ways. Such an analysis of performance characteristics is not only of interest for parameter tuning or hardware acquisition, but is also the basis for developing improved algorithms (cf. Bentes et al. [9]). Larsen et al. [10] perform an in-depth performance evaluation on different CPUs and a GPU of their unstructured grid volume renderer using data-parallel primitives, also comparing their work against alternative approaches. GPU-based rendering of particles has been an active area of research for many years as well [11], [12], [13]. In the following, we mainly focus on techniques that are used for rendering molecular dynamics visualization [14] (cf. Table 4).

A thorough analysis of performance is fundamental for the prediction and modeling of application performance in computer architecture and high-performance computing. Both, generic performance models [15], [16], [17] as well as approaches specific to a certain architecture (e.g., NVIDIA GPUs of the 200-series [18]) have been proposed. Such approaches are mostly based on variants of so-called microbenchmarks consisting of relatively small and specific pieces of code to test for certain characteristics. The models often target more or less generic compute applications, but there has also been some work focusing on scientific visualization, especially in the domain of high-performance computing. Rizzi et al. [19] presented an analytical model for predicting the scaling behavior of parallel volume rendering on GPU clusters, while Tkachev et al. [20] used neural networks to predict volume visualization performance. Larsen et al. [21] took a more general approach and modeled the performance of in-situ rendering in the context of high-performance computing. They investigated volume rendering, raytracing, and rasterization—three fundamental techniques for scientific visualization. Other proposed performance modeling and prediction methods focus on object-order rendering algorithms [22], [23], or on creating a performance model for a visualization pipeline [24].

3 Motivation of an Empirical Approach

While there are many different approaches on performance estimation (cf. Section 2), we focus on measurement-based estimation in this paper. In interactive scientific visualization, using empirical measurements to report performance is by far the most common approach. Interactive applications can typically generate many measurements comparably fast, requiring only little alterations to the code. This way, empirical measurements are able to capture the various factors influencing the performance of such applications.

In direct comparison with analytic performance modeling, which is another common approach, there are several advantages of an empirical approach in the case of interactive visualizations. First, concrete performance numbers achieved in practice are reported. However, using measurements instead of theoretic estimations also introduces uncertainty on the portability of performance numbers to other systems, data sets, and configurations. We address this uncertainty in part with this work by performing and analyzing extensive measurements. In general, there are cases in which the assessment of portable performance numbers becomes very hard or even impossible. With increasing complexity of a system, it becomes difficult to take all possible influencing factors and their combined performance characteristics into account. In those cases, an empirical approach is often better suited. Finally, the performance impact of some factors is hard or even impossible to assess without executing the algorithm. Early ray termination (ERT) as acceleration for volume raycasting is such an example. The performance influence of this optimization technique highly

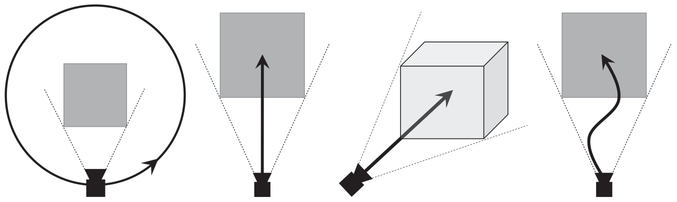


Fig. 1. Different camera paths used in measurements: orbit around data set, straight path along axes, diagonal, and sine curve.

depends on the data and the employed transfer function. Using ERT, even a small change to the transfer function can dramatically impact the number of samples along rays, e.g., if a surface changes from transparent to fully opaque. It would be very hard to cover such a change by a model, e.g., when estimating performance based on the numbers of samples and cost per sample.

4 Overview

In this paper, we discuss how the current practice of empirical performance evaluation in scientific visualization research compares to our approach of using extensive measurements. This includes investigating the question whether the evaluation criteria typically used in publications are a generally valid approach to report performance. Furthermore, we evaluate how we can possibly improve on the current practice to be more expressive. Note that in this work, we look at the currently common approach of performance evaluation for novel or improved interactive techniques in scientific visualization, also including techniques that improve the visuals. Our work is, however, not focused on comparing against performance evaluation techniques in particular.

Our basis is measuring an extensive amount of different parameter configurations and data sets on various systems for two distinct scientific visualization applications: volume raycasting and particle rendering. Both applications are custom implementations that are embedded in a benchmarking framework handling program flow, automatic parameter changes and logging of the runtime measurements. The framework and both applications are open source and can be accessed via https://github.com/UniStuttgart-VISUS/trrojan.

We focus on the performance of the two studied techniques in terms of frame times, which are crucial especially for interactive applications. However, there are other important performance measures as well, such as power and memory consumption. Furthermore, we restrict ourselves to scientific visualization applications in a single node environment that use GPUs without out-of-core-techniques. We presume though, that many of the concepts may be transferred to a broader field of visualization applications and usage scenarios.

A simple investigation of sequences of frame times (for instance, using percentiles) is not expressive for the immense number of measurements and resulting frame times we gathered. Therefore, our analysis approach is to use descriptive statistics, i.e., we mainly look at distributions, linear correlations of influencing factors, subsets of our data, and general patterns. Here, we do not investigate single outliers and specific measurement points.

TABLE 1
GPUs Used for Performance Measurements

Vendor	Model	Architecture	Shader units
AMD	Radeon R9 290*	GCN 2	2560
AMD	Radeon R9 Nano*	GCN 3	4096
AMD	Radeon RX 480	GCN 4	2304
AMD	Radeon Vega FE	GCN 5	4096
NVIDIA	GeForce GTX 980*	Maxwell	2048
NVIDIA	Quadro M6000	Maxwell	3072
NVIDIA	TITAN X (Pascal)	Pascal	3584
NVIDIA	GeForce GTX 1080 Ti	Pascal	3584
NVIDIA	Quadro GP100*	Pascal	3584
NVIDIA	TITAN Xp	Pascal	3840

**Only used in volume rendering benchmark.*

4.1 Experiment Design

For conducting our experiment, we developed an extensible benchmarking framework which allows us to automatically evaluate all combinations of influencing factors (rendering parameters, resolution, data set, ...) based on a simple declarative description. This description supports an arbitrary amount of Boolean or numerical values with up to four dimensions each as well as strings to to encode categorical factors such as file names of data sets. The factors that can be measured also include a series of pre-defined camera paths ranging from the commonly used orbits around the data set, over fly-throughs in all directions on straight or curved paths, to randomly selected views (cf. Fig. 1). Our framework provides a plugin mechanism, by means of which different kinds of benchmarking *environments* (here: compute APIs) and *devices* (here: GPUs) can be added. For our experiment, we implemented two environments, one for OpenCL 1.2 and one for Direct3D 11. The rationale behind those choices is that both APIs work for GPUs from different manufacturers. All GPUs that we tested in our experiment are listed in Table 1. For measuring their rendering performance, we used similarly equipped machines. These comprised an Intel Xeon E5-2630 processor running at 2.2 GHz and 64 GB of RAM per CPU for all GPUs except for the NVIDIA TITAN Xp (Intel Core i9-7900X at 3.3 GHz) and the GeForce GTX 1080 Ti (Core i7-7700K at 4.2 GHz). All systems ran on Windows 10 1709 (64-bit) with the latest stable drivers from AMD and NVIDIA, respectively.

4.2 Analysis Process

For analyzing our measurement data, we follow a top-down approach to gain insights into our large body of performance measurement data with little prior knowledge. Accordingly, we begin by looking at the overall distribution of the timings we obtained, investigating whether performance data of volume rendering and particle raycasting follow any known distribution. In a second step, we continue by investigating whether we can find a linear interrelationship between factors that we varied during the experiment. There are several reasons for focusing on linear dependencies: first, such results may be a convenient and intelligible way to describe the overall influence of a factor. Second, we would expect factors, such as the used hardware device, to have a linear influence on the results as long as all devices have the same capabilities. While we do not feature out-of-core rendering in our test applications yet, we would expect a non-linear performance impact. For instance, if one of the devices has not enough memory capacity and needs to switch to out-of-core rendering (data streaming) at a certain data set size while the others do not have to, a noticeable difference in the performance data would be expected. The device's rendering performance would decrease more or less linearly with increasing data set size, while at the point when switching to use out-of-core streaming, there would be a performance gap. We plan to verify this assumption with implementing and measuring out-of-core techniques in future work. In general, we first compare correlation coefficients on a per feature level, i.e., we look at the means and ranges of the correlation factors for the different specificities of each feature.

We proceed with looking into each of the influencing factors that we varied in more detail, by further investigating correlations. Here, we focus on single values of the respective factor, i.e., examining correlation matrices. By using linear regression, we additionally calculate concrete speed-ups for different factor values for the linear behaving factors that we determined before. We conclude the analysis process with a closer look at interesting findings and anomalies in the data.

5 Case Study 1: Volume Visualization by Means of GPU Raycasting

As our first case study, we examine volume visualization, one of the classic and most fundamental techniques in scientific visualization. Volume visualization is a computationally demanding application, especially for high resolution data sets from state-of-the-art scanners and simulations. Therefore, evaluating performance in volume visualization applications usually has been – and still is – a central aspect of analysis when extending techniques or developing new approaches in the domain. In particular, we consider GPU raycasting techniques, which are nowadays the de facto standard in volume rendering.

5.1 Common Practice

We first review publications with a strong focus on GPU volume raycasting and performance-related aspects to assess the common practice of performance evaluation in the field (Table 2). The majority of publications deal with real-time volume rendering in single node environments. After a review of their individual approaches for performance evaluation, we identified the following commonalities that are shared by a majority of the works.

Except Wu et al. [29], a single GPU was used in all papers (in almost all cases from NVIDIA). On average, around five data sets were evaluated. Transfer functions were unspecified in most cases, but are often depicted in figures. Across different papers, the size of the data sets ranged from less than one Megavoxel to several Gigavoxels (with the exception of Hadwiger et al. [25], who used a data set with more than one Teravoxel). In most cases, camera positions and parameters were not specified explicitly. Five papers used orbital rotations around one or three axes [27], [36], [39], [49], [50], while two papers employed pre-defined user interaction sequences [33], [38]. The size of the viewports were specified in most cases, with some exceptions [28], [31], [34], [41], [45]. In most cases, a single resolution was measured, with a pixel count of less or around one Megapixel (1024×1024). In

TABLE 2
Performance Evaluation in Recent Selected Volume Rendering Papers

Authors	Year	GPUs[d]	Viewports	Data sets	Views
Hadwiger et al. [25]	2018	N	1	8	–[a]
Magnus et al. [26]	2018	N	3	8	–[a]
Wang et al. [27]	2017	N	3	3[c]	rot. x,y,z
Jonsson et al. [28]	2017	N	–[a]	4[c]	–[a]
Wu et al. [29]	2017	2× N	1	6	–[a]
Sans et al. [30]	2016	N	1	4	–[a]
Zhang et al. [31]	2016	I	–[a]	7	–[a]
Ament et al. [32]	2016	N	1	6	–[a]
Bruder et al. [33]	2016	N	1	6	> 500[b]
Zhang et al. [34]	2016	N	–[a]	6	–[a]
Ding et al. [35]	2015	N	1	4[c]	–[a]
Sugimoto et al. [36]	2014	N	1	2	rot. x,y,z
Hero et al. [37]	2014	N	1	5	–[a]
Frey et al. [38]	2014	N	1	4[c]	> 1000[b]
Lee et al. [39]	2013	N	1	5	360
Liu et al. [40]	2013	N	2	9	–[a]
Yang et al. [41]	2012	N	–[a]	6	–[a]
Jonsson et al. [42]	2012	N	1	5	1 or 2
Kronander et al. [43]	2012	N	1	8	–[a]
Schlegel et al. [44]	2011	N	1	7	–[a]
Hernell et al. [45]	2010	N	–[a]	1	–[a]
Zhou et al. [46]	2009	N	1	4	–[a]
Lundström et al. [47]	2007	N	1	3	–[a]
Ljung et al. [48]	2006	A	1	4	2
Stegmaier et al. [49]	2005	N	2	1	rot. y
Bruckner et al. [50]	2005	N	1	1	3×360
Krüger et al. [51]	2003	A	1	4	–[a]

[a]*Property not mentioned in the paper.*
[b]*Interaction sequence with variable number of views.*
[c]*Time-series data sets.*
[d]*N = NVIDIA, A = AMD/ATI, I = Intel.*

some papers, several viewports were evaluated [26], [27], [40], [49]. Other rendering parameters were given to some extent, e.g., raycasting step sizes are specified or described in roughly half of the papers listed. Algorithmic variations, such as acceleration techniques and illumination methods, were stated often, but not always. In terms of performance indicators, most authors provided at least a single frame-per-second value per data set. Although it is seldom stated explicitly, we assume these numbers to be average values of multiple measurements. Some authors additionally provided minimum and maximum frame rates [49], [52], speed-ups or even frame time diagrams. Memory consumption is also reported in a few works.

In our first case study, we test the parameters usually evaluated in common practice: GPUs, viewports, data sets, and camera paths. We do not consider parameters that are specific to certain techniques such as illumination. We sampled each parameter dimension at least as extensively as in the respective papers. In addition to the parameters directly derived from common practice, we measured different sampling sizes along the ray, two acceleration techniques, and transfer functions. Testing various GPUs help in understanding device portability. Sampling rate parameters are numerical and have a well defined order, therefore providing means to test for scalability. Different data representations are covered by measuring data sets and transfer functions. With the camera setup being not reported in many of the

TABLE 3
Varied Parameters in Volume Rendering Benchmark

Factor	Number	Values
Device	10	see Table 1
Viewport	3	$512^2, 1024^2, 2048^2$
Step size	4	$0.25, 0.5, 1.0, 2.0$
Acceleration	2×2	ERT, ESS (on/off)
Camera path	7×36	$orbit_{x/y}, diagonal, path_{x/y/z/\sin z}$
Transfer function	2	see Fig. 2
Data set	21	see Fig. 2

reviewed papers, we wanted to test how different camera paths influence performance and what can be considered to be most representative. Evaluating with and without early ray termination and empty space skipping (ESS) gives us the possibility to emulate an accelerated rendering technique.

5.2 Our Test Implementation

We systematically analyze the performance of volume rendering, using our performance evaluation framework (see Section 4.1) and our custom implementation of GPU-based volume raycasting. For this, our design choices are directly derived from the reviewed papers to reflect the current standard approach in the field (cf. Section 5.1). Accordingly, we employ a front-to-back volume raycaster with perspective projection, which usually results in a higher thread divergence compared to orthographic projection. The renderer features local illumination based on gradients (calculated with central differences), and optionally, we also use two acceleration techniques: ERT and empty space skipping. For our implementation, we make use of the OpenCL 1.2 compute API, because it enables us to compare the exact same implementation across graphics cards from different vendors. Note that GLSL and CUDA are frequently chosen for implementation as well, yet the impact of these choices on performance is typically negligible. Volume raycasting is a comparably simple algorithm and its parallelization is trivial, which should theoretically minimize the advantage of vendor-specific APIs with respect to kernel execution times. We also do not use any specific (vendor optimized) libraries.

Table 3 lists the parameters that we varied in our volume rendering benchmark. In the following, we denote them as influencing *factors*. We further divide those factors into two classes: numerical (viewport and the step size along the rays) and categorical (the rest). We propose this distinction, because categorical factors do not have a well defined order, while we can sort numerical ones. The categorical factors also include the binary factors, such as the use of early ray termination. Besides variations of sampling resolutions in object space and image space, we used 21 different data sets for our evaluation. We use only two different transfer functions, because in combination with the use of acceleration techniques, distinct transfer functions can be seen as different data shapes (see Section 5.3 for a more detailed discussion). The step sizes in Table 3 are relative to the voxel length. That means for instance, a step size of 0.5 results in 2 samplings inside a voxel if the ray is parallel to one of the voxel's edges.

We ran our application on various discrete GPUs from NVIDIA and AMD, spanning multiple generations of different architectures. All of the GPUs used for the benchmark have sufficient memory to store the test data sets. To study the impact of simple performance improvement techniques, we enabled respectively disabled early ray termination and empty space skipping. We employ an image-order empty space skipping approach that is designed to be used in a single rendering pass, i.e., no rasterization is involved. For that, we implemented a 3D digital differential analyzer (DDA) to sample a low resolution representation of the volume data determining empty bricks and skipping them on-the-fly. The time needed for data pre-processing is not part of the measurement. The majority of the data sets we measured have also been used for performance evaluation in the reviewed papers (Table 2). Their resolutions range from $256^2 \times 128$ up to 1024^3 voxels. The spacing along the x- and y-axes is the same for all data sets, while the slice size along the z-axis differs for about half of the data sets. Notably, the tested data sets also include three generated cubic data sets: one with a uniform density value, another one with a density gradient from one corner to the opposite, and one featuring a high frequency pattern generated by using trigonometric functions. Exemplary renderings of all data sets but the uniform box are shown in Fig. 2. Although our implementation can handle and convert 8, 16, 32 and 64 bit data precision and sampling precision per voxel, we limit ourselves to 8 bit sampling precision in the following for the sake of better manageability of the data (every factor significantly raises the number of measurements due to the added dimension). In our pre-tests, they exhibited linear behavior for different data precision across most devices. The performance on AMD GPUs was affected a little less than on NVIDIA cards when increasing the scalar precision, with the exception of the AMD N9 Nano that was affected most of all cards.

To emulate different evaluation methods, we designed simple camera paths that can be sampled in arbitrarily small

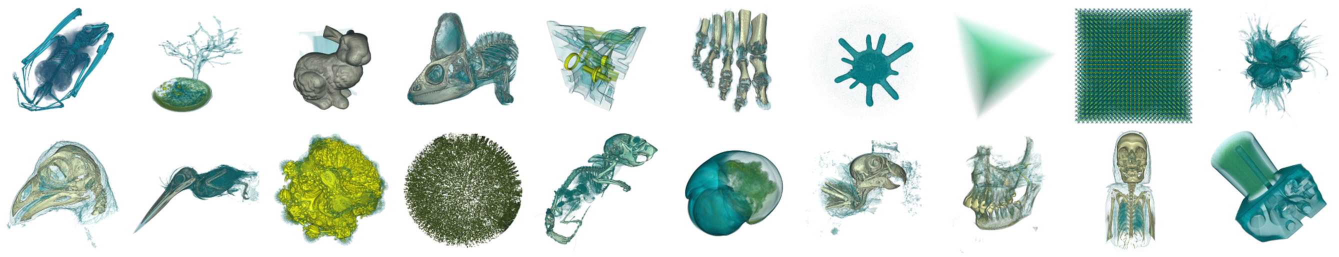


Fig. 2. Renderings of the tested volume data sets from CT scans, simulations, and artificial generation. From left to right: Bat, Bonsai, Bunny, Chameleon, Engine, Foot, Foraminifera, Gradient box, Frequency box, Hazelnut (top row); Hoatzin, Kingfisher, Mandelbulb, Porous media, Mouse, Supernova, Parakeet, Skull, VisFemale, Zeiss (bottom row). The uniform box is not shown.

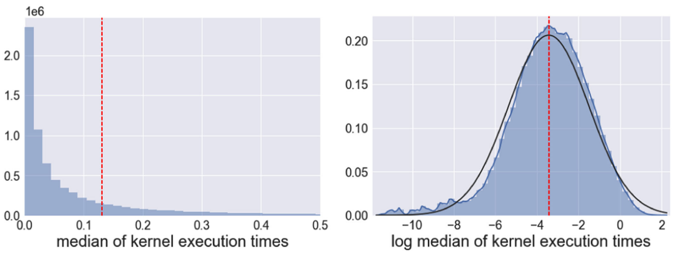


Fig. 3. Overall distribution of execution time medians (left) and scaled logarithmically (right) of the volume raycastings. Red lines mark the global mean, a log-normal distribution fit shows the deviation thereof.

intervals. To keep the computational effort manageable, we used 36 samples (camera positions) per path, adding up to a total of 252 different camera positions. The employed paths are schematically depicted in Fig. 1. We use a full orbit around the x-axis and y-axis (as in [27], [36], [39], [49], [50]), three straight paths into the volume data set along each axis, a diagonal path from one corner of the bounding box towards the opposite one, and one path along a scaled sine-curve in z-direction. All paths begin with a full view of the bounding box and (except the orbits) stop at its center. For the orbits, the view direction is always towards the center of the data set. We measured the kernel execution time of each configuration by using the queue profiling functionality of OpenCL, and rendered every configuration at least five times. In total, we conducted more than 25 million measurements with over five million distinct configurations.

5.3 Discussion of our Measurements

First, we look at the overall distribution of all kernel execution times of our volume raycaster (Fig. 3, left). This includes all measurements, but we took the median of the execution times for measurements with the same configuration. Differences between runs with the same configuration are generally very small (mean standard deviation of 0.002 s). Sometimes, the first run deviates slightly from the others (especially on AMD GPUs), which can probably be attributed to caching effects of the kernel. Applying a logarithmic scaling on the measured execution times reveals a log-normal nature of the distribution (Fig. 3, right). To verify this, we performed a one-sample Kolmogorov-Smirnov test on the data, which rejected the null hypothesis of the sample coming from a log-normal distribution ($D = 0.78, p < 2.2 \cdot 10^{-16}$).

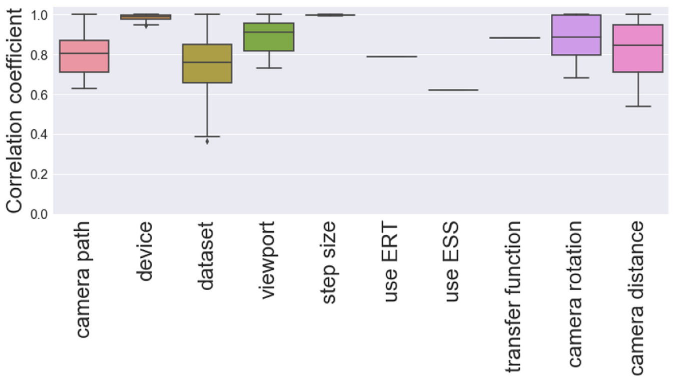


Fig. 4. Pearson correlation coefficients as boxplot for all measured factors and camera rotation/distance, influencing the volume rendering.

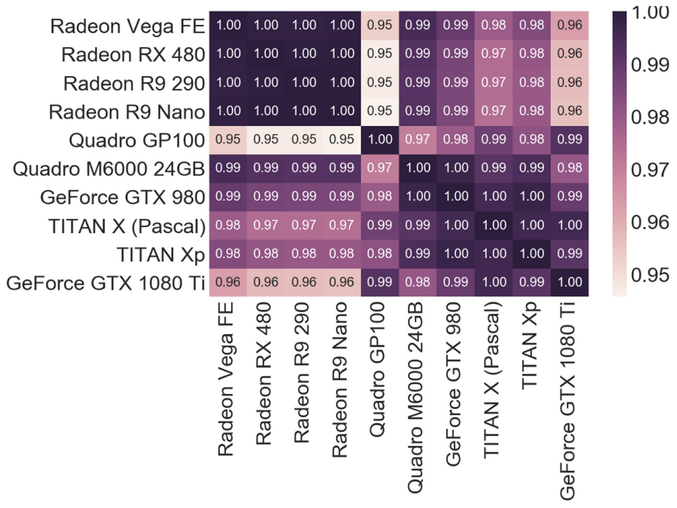


Fig. 5. Matrix showing Pearson correlation coefficients among different measured GPUs for volume raycasting.

On the one hand, there are some visible deviations in the tails (see Fig. 3), on the other hand, it is a known problem of the test becoming very sensitive to even small deviations for large sample sizes like ours (more than five million configurations). Although we cannot claim statistical support for the assumption that the performance of volume renderings is log-normal distributed, the visual inspection of the histogram shows that it is very close. The global distribution gives a general impression of the range and occurrence of the execution times. However, this representation does not provide any details about the influence of the different factors we varied in our measurements.

Therefore, we proceed with investigating the correlation of different factors with respect to the execution time (the target) with the other values/categories from the respective factor. For instance, we consider all timings obtained for the "NVIDIA TITAN Xp" being one data set and all for the "AMD Radeon Vega FE" being the other, and compute the Pearson correlation coefficient between these two disjunctive data sets. The result gives us an idea of whether there is a linear correlation between the rendering performance of the two devices. If that would be the case, we could conclude that the devices behave similarly for all test cases except for a linear scaling factor and offset. We compute such a correlation for all pairs of different values of a single factor, which yields a correlation matrix (e.g., Fig. 5) and repeat this procedure for every factor we tested, i.e., we calculate one such correlation matrix per factor. Fig. 4 gives an aggregated overview of all those matrices: each single box in this boxplot represents the distribution of all correlation factors within one matrix. This yields one box per tested factor. That is, high values in this diagram indicate that there is generally a high linear correlation between all pairs of levels of the respective factor. We also calculated separate correlation matrices for camera rotation and camera distance, which we added to the boxplot as well to compare against the camera path factor.

Looking at the means of the correlation coefficients, we see that the employed volume data set has the highest variance and lowest correlation calculated (also the factor with the highest number of different values in our measurement), followed by the camera path. While the step size has a very

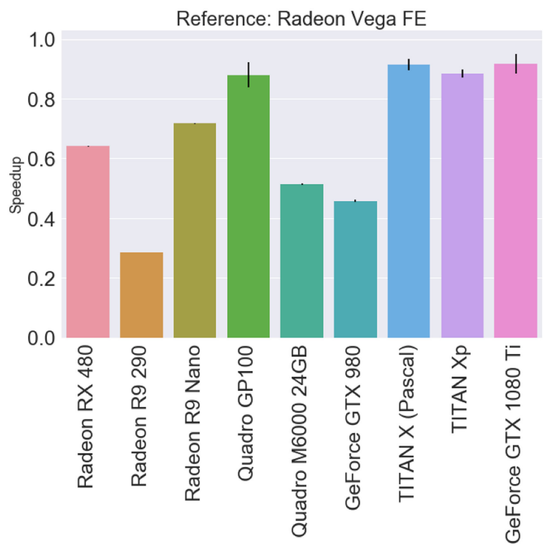


Fig. 6. Speed-up for volume raycasting on different GPUs relative to the AMD Radeon Vega FE. Error bars indicate uncertainty.

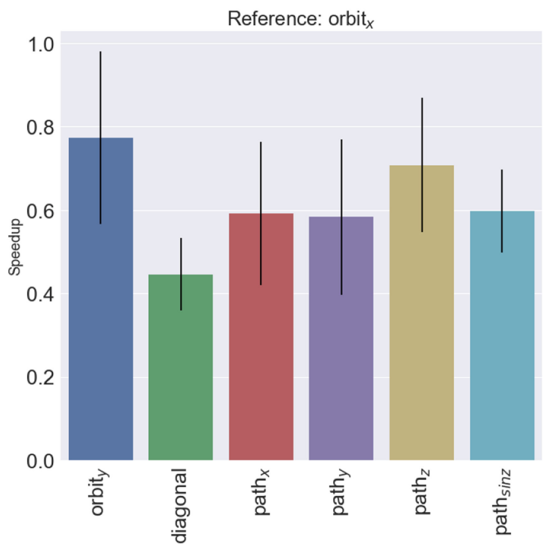


Fig. 7. Speed-up for volume raycasting using different camera paths relative to the orbit around the x-axis. Error bars indicate uncertainty.

high correlation among different values, there is also some noticeable variation for the viewport. We attribute this effect to performance deviations caused by oversampling of the smaller data sets (i.e., neighboring rays get very cheap due to caching effects). The binary factors for our acceleration techniques and transfer functions have a medium to high correlation. They are closely linked to the data set structure. Interestingly, all devices show a very predictable behavior in terms of performance.

For investigating the correlation factors more closely, we can plot the correlation matrix as a heat map. Fig. 5 shows such a correlation heat map for the *device* factor, i.e., the different GPUs. Notably, the linear correlations between the tested AMD GPUs are almost 1.0 in all cases, while the correlations with the NVIDIA cards are still very high, with factors above 0.95 in all cases.

The factors can further be classified into four categories. The first one being the hardware (in our case GPUs), the second one the numeric sampling parameters in image space and object space (viewport resolution and step size along the ray), and the third one the camera parameters, which are covered by the different paths in our measurements. The four remaining factors can be combined to a category defining the data structure, because the data set combined with a transfer function may be seen as just another data structure. One may also argue that different camera parameters are also a form of distinctive data representations, respective structures. That means, by changing the camera perspective, we "generate" a new structure, through processing only parts of the original data and changing the behavior/impact of ERT and ESS. Therefore, a reduction of the four categories to three (adding camera to data structure) is also possible.

In terms of those categories, the determined correlation coefficients imply that using different hardware results in comparably predictable behavior as well as the sampling factors (with some limitations in the form of the viewport). The camera parameters show significantly bigger deviations, while factors related to the volume data structure have the lowest correlation among one another. Therefore, we conclude that the data set structure is the most important factor for performance quantification in our volume raycasting application.

To validate this conclusion, we separate the data based on all factors related to the volume data structure as discussed above: data set, transfer function, acceleration techniques and camera. We then apply a simple linear regression on a training subset of the data. When testing the resulting model, we achieve an average coefficient of determination of $R^2 = 0.894$ across all data sets, with a minimum of $R^2_{min} = 0.756$ and a maximum of $R^2_{max} = 0.940$. In order to generate a reference, we used random forest regression. Using this more advanced machine learning technique, we were able to achieve a coefficient of determination of $R^2 = 0.982$. However, it is not the objective of this paper to accurately predict execution timings. We deliberately chose the simple linear regression to show and to understand simple relations in the data, which is not necessarily possible with advanced machine learning techniques, such as random forests or neural networks. Figs. 6 and 7 show the linear relation for all tested GPUs as well as camera paths. Here, the error bars indicate uncertainty and are calculated using $error = (1-r) \cdot s$, with s being the slope of the regression and r the correlation coefficient.

Based on these findings, we were interested in two additional aspects: a detailed look into the volume data sets, which showed the highest variation among all features, and the influence of the camera parametrization on performance.

To accomplish the former, we measured the performance of a stack of down-sampled data sets (i.e., aggregating $2^3, 2^4, 2^8, \ldots$ neighboring voxels). As an example, we used the Chameleon data set with an original resolution of 1024^3 voxels and created seven down-sampled variants, the smallest one having a resolution of 128^3 voxels. Investigating the correlation matrix of those data sets showed a minimum coefficient of 0.85 (between the highest and the lowest resolutions), with a mean value of 0.97 for all combinations. Another example using the Zeiss data set with an original resolution of 640^3 and six down-samplings showed similar results, with a mean correlation coefficient of 0.99 a minimum of 0.96. These results suggest that the structure of the

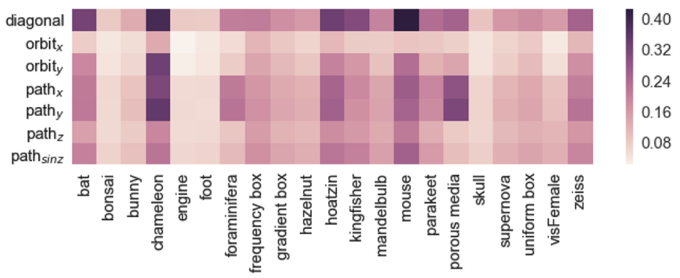


Fig. 8. Mean frame time (seconds) for rendering the volume data sets, using different camera paths.

data set is mainly responsible for the variance in performance behavior. However, there might be cases where a high resolution data set has distinct performance characteristics from a down sampled version of the data. For instance, this could be the case if fine structures or noise in the original data get averaged out in a down-sampled version and therefore result in faster rendering times.

Fig. 8 shows a heat map for the execution time means of all data sets when using a specific camera path. It shows that the performance deviation are comparably high for several combinations, which motivated us to further investigate the performance behavior for different camera parameters. Therefore, we calculated Pearson correlation coefficient matrices for solely the rotation respectively the distance to the center of the bounding box of the volume data set. For the former, we used the samples from the camera path doing a full orbit around the y-axis. For the latter, we took the samples of the straight camera paths along the three major axes. Different positions among the orbit showed a mean correlation of 0.88, with the minimum being 0.68 (cf. Fig. 4). Besides the structure, shape, and spacing of a data set, memory access patterns and caching influences performance during rotation around a data set in texture-based volume rendering [27], [36].

For the distance to the volume (i.e., zoom into the data set), the mean of the correlation coefficients was lower compared to the one for rotation, at 0.83. Notably, the lowest factor was 0.54: it is the correlation factor between the camera configuration with the shortest distance to the center and the configuration with the highest distance. Overall, correlation to other camera configurations decreases with decreasing distances to the center. This suggests a higher divergence in performance behavior between zooming into a volume than circling around it.

We also performed several two-sample, two-sided Kolmogorov-Smirnov tests to analyze whether the intuitive choice of rotating the camera around the data is representative for a wider range of views in the sense that the sample comes from the same distribution as all views. For both orbit paths, the test rejected the null hypothesis ($D = 0.09, p < 2.2 \cdot 10^{-16}$ for $orbit_x$ and $D = 0.04, p < 2.2 \cdot 10^{-16}$ for $orbit_y$) whereas it rejected it for none of 100 randomly chosen samples of the same size (approx. 1.1 m), with p-values ranging between 0.051 and 0.997 (mean 0.644, median 0.686).

6 Case Study 2: Particle Visualization

In our second case study, we examine particle rendering in the form of several different techniques of raycasting sprites.

TABLE 4
Performance Evaluations in Recent Selected Particle and Molecular Rendering Papers

Authors	Year	GPUs[b]	Viewports	Data sets	Views
Müller et al. [53]	2018	H	1	5	$5 \times ?$[c]
Ibrahim et al. [54]	2018	N	1	6	2
Hermosilla et al. [55]	2017	N	1	9	512
Jurčík et al. [56]	2016	N	1	4	1
Skånberg et al. [57]	2016	N	1	4	1
Grottel et al. [58]	2015	N	–[a]	3	–[a]
Wald et al. [59]	2015	4×I	1	7	2
Guo et al. [60]	2015	4×N	1	2	7
Knoll et al. [61]	2014	2×I, N	1	8	2
Le Muzic et al. [62]	2014	N	1	1	1
Grottel et al. [63]	2012	N	1	6	1
Lindow et al. [64]	2012	N	1	7	rot.
Chavent et al. [65]	2011	4×N	1	12	1
Lindow et al. [66]	2010	2×N	1	5	–[a]
Grottel et al. [67]	2010	N	1	5	4
Krone et al. [68]	2009	N	1	10	1
Falk et al. [69]	2009	N	1	6	2
Lampe et al. [70]	2007	N	1	6	–[a]
Gribble et al.[71]	2007	1×O	2	6	1
Tarini et al. [72]	2006	1×A	–[a]	≥ 2	–[a]
Reina & Ertl [73]	2005	3×N	–[a]	4	≥ 2
Klein & Ertl [74]	2004	N	1	1	–[a]
Gumhold [14]	2003	N, A	–[a]	1	–[a]

[a]*Property not mentioned in the paper.*
[b]*A = ATI/AMD GPU, H = HoloLens, I = Intel many-core CPU or Xeon Phi, N = NVIDIA GPU, O = Opteron CPU.*
[c]*Multiple camera paths with variable number of frames.*

Those techniques are often used for visualizing particle-based data sets such as molecular dynamics simulations.

6.1 Common Practice

Table 4 shows a summary of the evaluation techniques used in publications on particle-based visualization over the last decade. It can be seen that the authors mostly focused on varying the data sets, while in most cases only one GPU and one viewport size were tested. Information about the camera was rarely mentioned, especially in older publications. More recently, the camera was oftentimes adjusted such that the data set fits the available screen area [56], [63], [68]. Other authors complemented this overview rendering with a close-up [62], [69], [73], sometimes including the actual renderings for said views for reference [59], [61], [71]. In three cases [62], [64], [67], the authors used knowledge about their algorithm and placed the camera in the assumed worst-case position. Performance numbers for a larger number of camera positions are rarely reported. If so, rotations around the data sets are used [64] that can also be complemented by fly-through paths [53]. Hermosilla et al. [55], as an exception, report averaged frame rates from random camera positions on a sphere around the data set.

The predominant measure used in basically all cases are frames per second, as in the volume rendering case. An exception is the work of Gumhold [14], who reported for a series of different data set sizes the number of ellipsoids rendered per second along with the number of fragments filled per second. Frame rates were sometimes complemented by detailed performance information regarding individual steps of the algorithm [54], [60], [61], [66], [67], [68]. These were typically

TABLE 5
Techniques and Required Shader Stages Used for Particle Rendering

Technique	Active shader stages*	Unique factors
Screen-aligned quad	VS, **GS**, PS	
Ray-aligned quad	VS, **GS**, PS	
Instanced quad	**VS**, PS	color conversion
Ray-aligned quad	VS, **HS, DS**, PS	
Ray-aligned polygon	VS, **HS, DS**, PS	corners (4–8, 16, 32)
Adaptive polygon	VS, **HS, DS**, PS	allowed corners (4–16)

**Bold face marks the shader stage used to compute the sprites.*
V = Vertex, G = Geometry, H = Hull, D = Domain, P = Pixel shaders.

reported in milliseconds. Relative speed-ups [66], [71] were less common and normally given if a new technique was compared to an existing one [56], [58], [61], [64], [65], [70]. Besides timings, bandwidth [67], [70] and memory requirements [59], [61], [64] were reported from time to time. Müller et al. [53] also included data from pipeline statistics queries like the number of shader invocations or the geometry load.

While some authors provided quite detailed explanations about certain performance characteristics and their causes [59], [61] or compared several variants of their technique [57], [63], [68], [69], only the works of Grottel et al. [67] and Müller et al. [53] presented systematic performance studies. The former investigated different techniques to transfer data from main memory to the GPU, data quantization, two culling techniques and deferred shading, and the combinations thereof. The reported results did not only include frame rates but also statistics such as the visible data after culling. The latter concerned themselves with comparing the performance of different shader-based methods of rendering spherical glyphs on *Microsoft HoloLens*.

The parameters tested in the assessed common pratice are largely similar to the volume rendering case: GPUs, viewports, data sets, and camera paths. Among others, this allows us to compare across experiments. As there are many different techniques and variants used to perform particle rendering, we focus on evaluating them to help understanding the differences. Additionally, we test rendering specific parameters such as using conservative depth.

6.2 Our Test Implementation

We based our test implementation on the Direct3D 11 API, because the predominant technique for rendering particle data sets as spherical glyphs nowadays is computing the ray-sphere intersections on sprites, which requires an API supporting rasterization. In our experiment, we evaluated several variants of this GPU-based raycasting, which are implemented via modified pixel shaders. Our tests include a series of techniques for generating the sprites. Table 5 shows an overview of all those rendering techniques along with specific factors that might influence the rendering performance of the technique. They differ in the usage of quads or polygons, the alignment of quads, the use of instancing, the active shader stages, and the shaders used for computing the sprites. The instanced-based approach differs from all others in that it does not obtain its data (position, radius, and color of the particles) from a vertex buffer. Instead, data is obtained from a structured resource view, which is accessed based on the instance ID. The vertices for the instances are not stored in a vertex buffer, but computed on-the-fly from the vertex ID. The ray-sphere intersection itself is always computed in the pixel shader. For all tests using particles without a color, but with an intensity value, we also tested the transfer function lookup in the vertex and pixel shaders regardless of the technique. Additionally, we tested conservative depth output enabled and disabled for all techniques. Conservative depth output allows the GPU to perform early z-culling even as we are writing the correct depth of the ray-sphere intersection point computed in the pixel shader. This way, many fragments in dense data sets can be discarded before invoking the pixel shader. However, the effect varies with the position of the camera because the order in which particles are emitted depends on the fixed layout of the vertex buffer, not on the view. Therefore, the amount of overdraw and the order in which it happens – and in turn the number of fragments discarded early – are view-dependent.

As in the volume rendering case, we used quadratic viewports of 512^2, 1024^2 and 2048^2 pixels. The camera was moved in ten steps along different paths, namely $diagonal_{x/y/z}$, $orbit_{x/y}$, $path_{x/y/z}$ and $path_{\sin x/y/z}$. We measured the particle rendering on the same systems listed in Table 1 that we used for the volume rendering, except the ones marked with an asterisk.

Our tests included five real-world data sets (cf. Fig. 9) and 21 artificial ones with uniformly distributed particles in a 10^3 bounding box. The latter form mainly two series: the first one contains $10^k, k \in \{3, 4, 5, 6\}$ particles, each having approximately the same fraction of the bounding box filled (i.e., the size of the particles decrease as the number of particles increase). The second series always comprises 100,000 particles but their size increases over three steps, i.e., more of the bounding box is filled increasing the overdraw that occurs during raycasting. All artificial data sets were tested with RGBA8 and RGBA32 coloring and intensity only.

For each test configuration, we obtained mainly two data points: GPU timings measured with a timestamp query injected into the command stream and the wall clock time measured with the high-resolution timer on the CPU. The rationale behind this is that we wanted to compare the

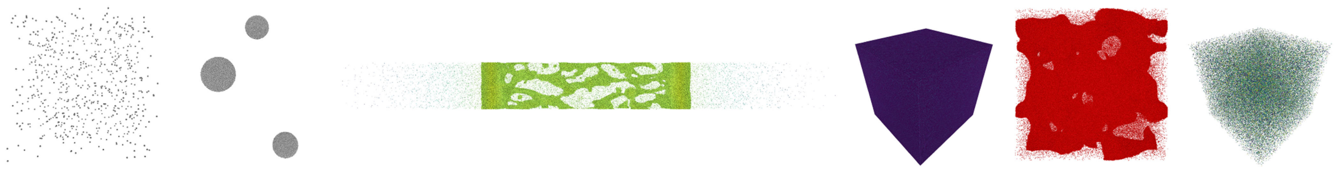
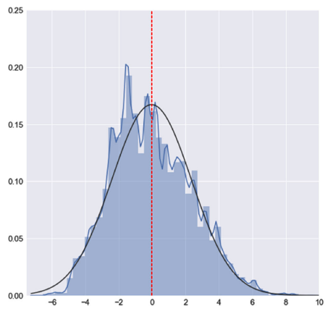


Fig. 9. The particle data sets from molecular dynamics simulations used in our tests. From left to right, these are a small test run (1,000 particles), three droplets (79,509 particles), the formation of a liquid layer (2,000,000 particles), a laser ablation (6,185,166 particles) and 10,000,000 particles. The last image shows one of the randomly generated data sets with 100,000 particles.


(a) Time stamp queries

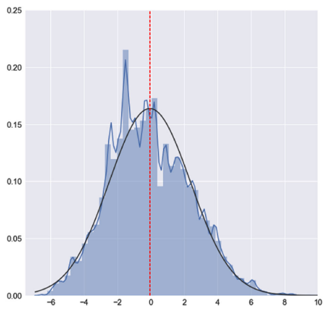

(b) Wall clock time

Fig. 10. Distribution of the logarithmically scaled timestamp queries (a) and wall clock times (b), of the particle renderings. The red lines mark the mean, whereas the black lines show a fitted log-normal distribution.

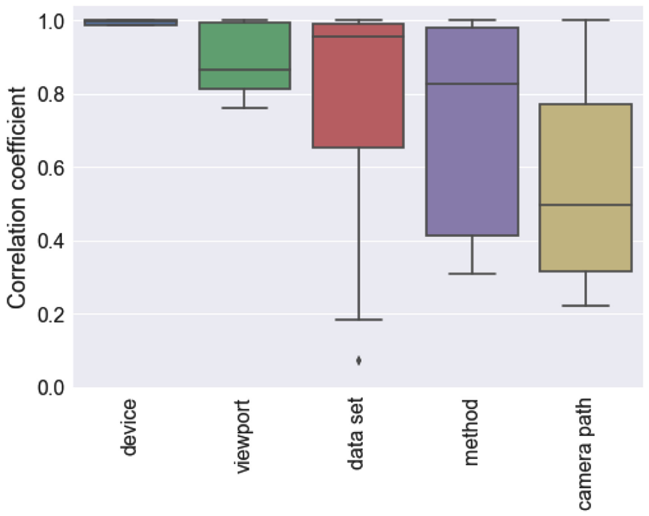


Fig. 11. Means of Pearson correlation coefficient for five selected factors influencing the rendering speed of the particle data sets.

timestamp queries, which measure very short periods of time, with the averaged CPU timings, which should expose less variation. Additionally, we obtained the number of shader invocations for each configuration by issuing a query for pipeline statistics. Before the actual test run, we performed at least four pre-warming renderings, which are not included in the result. These served mainly two purposes: the first frame after switching shaders is usually particularly slow and therefore should be excluded from the result for not being representative. At the same time, the system computed from the pre-warming renderings how many frames need to be rendered such that the wall time measurements cover at least 100 ms. The actual measurements were then performed in three separate steps: first, we rendered eight frames and obtained the timestamp query results for them. Unless denoted otherwise, the GPU times reported subsequently are the medians of these eight measurements. The difference between the minimum and maximum time for one measurement ranges between 0 ms and 1,385 ms (mean 0.16 ms). Second, we obtained the pipeline statistics by rendering the same configuration again. And finally, we rendered the number of frames computed beforehand to measure the wall time. As we completely rely on rendering to off-screen targets, which lacks the buffer swap as synchronization point, we stalled the CPU by waiting for an event query injected into the command stream after the last frame. Wall clock times in the following are the time between the first frame and the point when the event query returned, divided by the number of frames rendered during this period.

6.3 Discussion of our Measurements

We begin by looking at the histogram of GPU (Fig. 10a) and CPU (Fig. 10b) timings from all measurements. The applied logarithmic scale shows that the distribution vaguely resembles a log-normal one, but the fit is not as good for the volume rendering case. Given the histogram and the sample size of more than 3.3 million measurements, it is not surprising that a one-sample Kolmogorov-Smirnov test rejected log-normal distribution ($D = 0.25, p < 2.2 \cdot 10^{-16}$). Fig. 10 also shows that there are small differences in the distribution of GPU time stamp queries and wall clock measurements, but both histograms have approximately the same spikes and a similar mean. These spikes could either result from particle rendering generally not being log-normal distributed or from our test cases not sampling the parameter space sufficiently well. We reckon that the selection of the data sets is an important factor here, because we mainly sample based on orders of magnitude and not evenly distributed numbers of particles. Data set sizes of 100,000 particles are over-represented in the results and account for more than half of the observations in Fig. 10, because the series of tests for different sphere sizes uses this number of particles. The other artificial data sets comprising 1000 and 10,000 particles further add to a large imbalance of data sets below 1 million particles, which has a notable influence on the shape of the histogram.

Akin to the analysis of the volume rendering data, we investigated the mean correlation between the different levels of factors, such as the device, the viewport, etc. (cf. Fig. 11). Again, we see that there is a strong linear relationship between all devices, i.e., the behavior of the GPUs is generally the same for all tests. The correlation between the data sets is generally higher than for the volume rendering case, but there is a lot of variation and there are some outliers: both effects can be explained by the large number of artificial data sets in the test. The performance behavior of these seems to be largely the same with respect to the other factors with correlation factors close to one. One notable exception are the data sets with very small radii, which result in particles being represented by a single pixel in the majority of views. Their correlation factor with the other artificial data sets is only around 0.72 (cf. Fig. 12). The data sets from simulations naturally exhibit more variation, most notably the 79,509 particles that form three drops and therefore differ from all other data in that a large fraction of the bounding box is actually empty. This data set has a correlation factor of around 0.43 with most other data sets, except for the laser ablation (0.18) and the liquid layer formation (0.05). The latter has a Pearson correlation of around 0.39 with most other data sets and is special in the sense that it is the only data set with a strongly non-cubic bounding box, i.e., the path along the z-axis is much longer than along the other two. The two factors showing the least correlation on average, and in turn the greatest variance, are the rendering method and the camera path. While the methods based on sprites aligned with the view ray mostly have correlation factors above 0.9 among each other, the screen-aligned one lies around 0.7 depending on the method compared. The camera paths (Fig. 13) exhibit

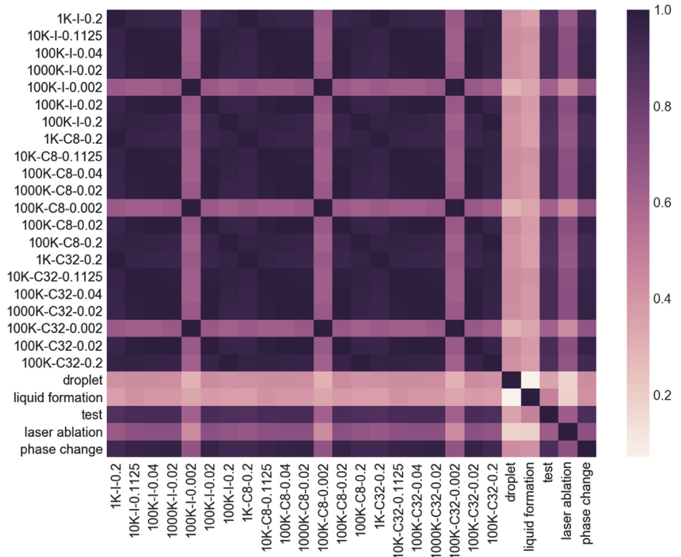
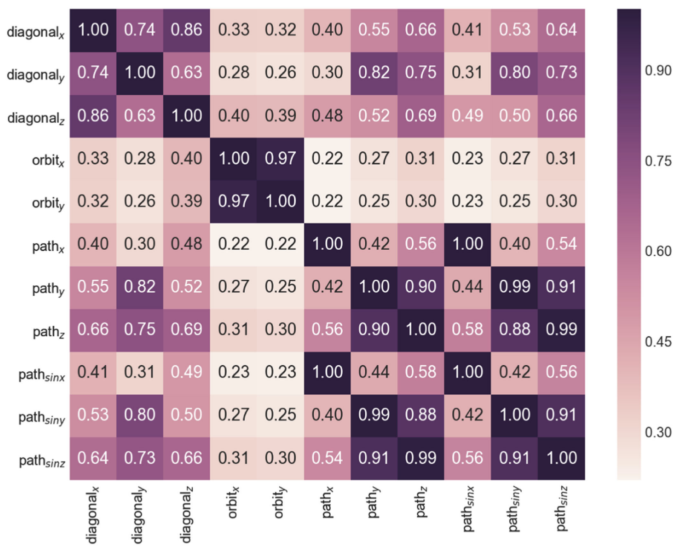


Fig. 12. Matrix of the Pearson correlation factors between timings for different particle data sets. Numbers omitted for better readability.

similar results: the paths along the axis (straight and sine) reach 0.96 or higher, while the two orbits yield around 0.9 with the other methods (comparing the orbits yields a 1.0). We found that the underlying root cause for both of these two observations is that the screen-aligned sprites can cause significantly more overdraw if the camera is close to a sphere. This can be measured via the number of pixel shader invocations that is an order of magnitude above the one for the ray-aligned sprites. While the ray-aligned quads can be clipped against the front plane (causing visual artifacts), the screen-aligned ones become larger as a sphere comes closer and are only clipped at once if they reach the front plane. Furthermore, depending on the radius of the sphere, the camera position and the clipping planes, the method might also generate sprites for back faces of spheres, which generate no fragment in the end. Obviously, this effect cannot occur if the camera is outside of the bounding box, which is the case for the orbit paths.



	$diagonal_x$	$diagonal_y$	$diagonal_z$	$orbit_x$	$orbit_y$	$path_x$	$path_y$	$path_z$	$path_{sinx}$	$path_{siny}$	$path_{sinz}$
$diagonal_x$	1.00	0.74	0.86	0.33	0.32	0.40	0.55	0.66	0.41	0.53	0.64
$diagonal_y$	0.74	1.00	0.63	0.28	0.26	0.30	0.82	0.75	0.31	0.80	0.73
$diagonal_z$	0.86	0.63	1.00	0.40	0.39	0.48	0.52	0.69	0.49	0.50	0.66
$orbit_x$	0.33	0.28	0.40	1.00	0.97	0.22	0.27	0.31	0.23	0.27	0.31
$orbit_y$	0.32	0.26	0.39	0.97	1.00	0.22	0.25	0.30	0.23	0.25	0.30
$path_x$	0.40	0.30	0.48	0.22	0.22	1.00	0.42	0.56	1.00	0.40	0.54
$path_y$	0.55	0.82	0.52	0.27	0.25	0.42	1.00	0.90	0.44	0.99	0.91
$path_z$	0.66	0.75	0.69	0.31	0.30	0.56	0.90	1.00	0.58	0.88	0.99
$path_{sinx}$	0.41	0.31	0.49	0.23	0.23	1.00	0.44	0.58	1.00	0.42	0.56
$path_{siny}$	0.53	0.80	0.50	0.27	0.25	0.40	0.99	0.88	0.42	1.00	0.91
$path_{sinz}$	0.64	0.73	0.66	0.31	0.30	0.54	0.91	0.99	0.56	0.91	1.00

Fig. 13. Matrix of the Pearson correlation factor between the timings for different camera paths through the particle data sets.

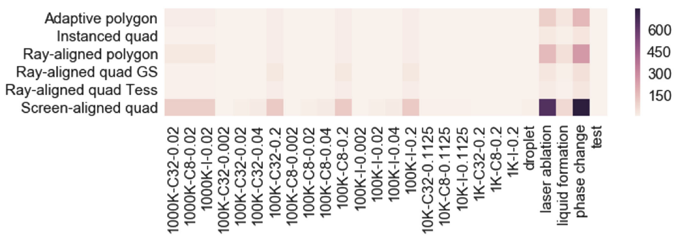


Fig. 14. Mean frame time (in milliseconds) for the particle data sets rendered using different sprite-based sphere rendering techniques.

We assumed that the radius of the spheres plays an important role for this effect, we further investigated the rendering times of all methods depending on the data sets. Fig. 14 reveals that this is actually the case: the high overdraw of screen-aligned quads becomes only a relevant factor if the number of particles is high or, in case of the artificial data sets, if the radius is—compared to real-world applications—unnaturally high. The matrix also shows that the generation of view-aligned quads performs almost the same for all data sets, only if the number of particles becomes very large, tessellation-based methods become slightly slower.

Given this observation, it is clear that an orbit path cannot be representative for the entirety of the views. We performed a two-sample Kolmogorov-Smirnov test, which rejected that both orbit paths have the same distribution as population of all views ($D = 0.03, p < 2.2 \cdot 10^{-16}$ for $orbit_x$ and $orbit_y$). Again, we took 100 random samples of the same size (more than 300 k), for most of which the test did not reject the null hypothesis of the same underlying populations. The p-values ranged from 0.017 to 0.999 (mean 0.605, median 0.658).

7 Discussion and Lessons Learned

One rather obvious conclusion we draw from our experiments is that specifying minimum and maximum values as well as percentiles along with average fps or frame times is desirable, although seldom done in practice. For instance, when performing a fully accelerated volume raycasting of the “Mandelbulb” on an NVIDIA TITAN Xp, the mean of 0.0456 s implies a decently interactive frame rate of 21 fps, but the 75th percentile is actually 0.0746 s and the 90th percentile only 0.1227 s. However, none of the reviewed papers give percentiles, many of them only report average fps.

Given the amount of measurements we obtained in two different areas of visualization, we believe it is reasonably safe to conclude that the typical approach of testing only one device (done by $\approx$83 percent of the reviewed papers) seems to be largely valid for most cases. There might be exceptions in case of different device types (CPU versus GPU) or disruptive technology changes enabling different kinds of algorithms. However, we found a linear behavior even across different memory technologies, architectures and vendors for the same algorithm.

Authors from both application areas oftentimes use different data sets for their evaluation: $\approx$89 percent in the reviewed volume rendering papers and $\approx$87 percent in the particle rendering ones. In our studies, we could confirm that this is important due to their generally significant impact on performance. However, in the case study on particle rendering, we found that measurements with artificially generated data sets must be handled with care. Real-world data usually have complex internal shapes influenced by many parameters, making it difficult to decide which parameters should be used in which way to generate test data. Furthermore, designing data sets with a specific property in mind that is to be tested, can cause an undesired bias in the distribution of the results. In hindsight, we would recommend separating the corpus of data sets for describing general performance characteristics and the ones for closer investigation of a priori known effects. However, more realistic performance characteristics when using artificial data could possibly be achieved with generative data models [75]. By using them, a potential scarcity of data sets could be circumvented, there is even the possibility of finding a small set of generated data sets that represents most of the common performance characteristics.

The camera parameters proved to have an equally high impact in both of our tests. However, evaluations in papers oftentimes only use a low number of view points. About 58 percent of the papers that actually report the number of views (which are only half the papers) use less than eight different camera configurations. Therefore, it is particularly desirable that a systematic performance evaluation explicitly states the respective configuration, and that a variety of different camera configurations are considered. Almost all reviewed papers that report on evaluating more than eight views perform some form of intuitive camera paths, e.g., rotations around the bounding box or recorded user interactions. We found that using those intuitive choices might be insufficient. Insofar, it might be sensible to test a number of randomly chosen camera poses—filtered in a way that a reasonable portion of the data is visible. Although random camera parameters are not a realistic usage scenario, using them for measurement has the advantage that the overall distribution and performance characteristics can be covered with less samples, thus freeing resources for measuring other parameters. This is also an area in which we intend to continue our work: measuring randomly chosen, but meaningful views to determine whether this has an influence on the overall distribution of rendering times and how many views are required to obtain a representative set of measurements. Another way to improve performance evaluation in this area would be an empirical collection of typical camera paths for specific application areas. A public repository of such common interaction sequences could serve as input to improve the evaluation.

7.1 Best Practices

From our results, we derived the following set of best practices for the performance evaluation of interactive scientific visualization techniques in single node environments:

- Report performance distributions instead of a single frames-per-second value (or report at least frame rates for different percentiles).
- Report all important factors that have an influence on the performance. Our literature review shows that factors like the viewport size seem to be so obvious to the authors that they are often missing in the exposition.
- Generally, one system for testing seems sufficient to report performance properties, if the software is not tailored towards special hardware characteristics like specific memory hierarchies.
- Multiple (meaningful) random camera configurations should be used for measurements. While these might not be representative for the actual interactions of a user in a specific scenario, they allow for a comprehensive general assessment. Typically, they yield much more expressive results in comparison to using just a single view point or a single path.
- Optimally, a large variety of (real-world) data sets and/or different shape-defining properties (such as transfer functions in volume rendering) should be measured. This is particularly important when acceleration techniques are used, as their performance results typically strongly depends on data characteristics.

Note that this list is only a rough guideline addressing several properties that we discovered during our data analysis. Although we believe that they are a good starting point, the importance and portability may vary depending on the specific visualization application and domain. We consciously tried to keep these recommendations as general as possible, while also providing a guideline on how to improve performance evaluation.

7.2 Limitations

Besides the still limited number of camera poses we have tested, there are several limitations to our current approach: we restricted all of our measurements to interactive algorithms running on a single GPU and no out-of-core handling. Furthermore, we collected measurements for only one dimension—the rendering times—while there are many other dimensions that might be interesting or necessary for a detailed evaluation. For instance, the Direct3D pipeline statistics with the pixel shader invocations proved to be very helpful for interpreting certain effects in the results from the particle test runs.

Given the sheer amount of data, we followed a top-down exploratory approach, but investigating correlations between factors is still at such a high level that potentially interesting individual outliers are hard to find. Also, we only analyzed whether there is a linear or no correlation, which is sufficient to describe the influence of a factor in principle and to derive some guidelines on how to handle this factor. However, we cannot find out whether only certain expressions of a factor influence others this way, e.g., if an optimization has only an impact on a subset of the data sets or in other specific parameter configurations.

8 Conclusion and Future Work

We investigated the current approach on performance quantification in scientific visualization by reviewing respective efforts in related work and comparing these methods to an extensive measurement approach. Thereby, we evaluated volume rendering and particle rendering, two established fields in scientific visualization, in which the commonly employed basic algorithms feature distinctive properties. An in-depth analysis of our measurement data, which comes from the—to our knowledge—largest systematic series of performance measurements of different scientific visualization applications, showed several characteristics and commonalities between the two cases. Based on those we are now able to answer the questions asked in the introduction, to some extent: Which are the most important performance factors, how big should a subset size be, and are there typical correlations of factors? We summarize our findings in a list of best practices, that can be seen as a guideline for empirical performance evaluation of interactive scientific visualization techniques.

We believe that our work is just a first step towards a better understanding of how the performance of scientific visualization applications behaves at large and therefore think it opens up multiple directions for future work. Foremost, we plan to extend our approach to parallel and out-of-core techniques of the two use cases and other domains in visualization beyond volume rendering and particle rendering. Overall, the goal is to obtain a more general understanding of performance quantification. We also want to extend our current measurement series based on the insights we gained so far. While we focused on frame times as a performance metric in this work, there are many others that we would like to consider in future work including, memory consumption, energy usage, rendering quality, or even user satisfaction.

A key part of experimental measurements is the reproducibility of the results, i.e., others should be able to redo and validate them. We also think that it is important to share measurement data, so that other researchers have the opportunity to explore it and possibly gain new insights. Therefore, our benchmarking framework as well as our measurement results are available at http://trrojan.visus.uni-stuttgart.de. We plan to continuously expand the data base, for instance, with measurements of additional camera positions, hardware, and other visual computing applications.

ACKNOWLEDGMENTS

The authors would like to thank the German Research Foundation (DFG) for supporting the project within the projects A02 and INF of SFB/Transregio 161.

REFERENCES

[1] R. S. Laramee, "How to write a visualization research paper: A starting point," *Comput. Graph. Forum*, vol. 29, pp. 2363–2371, 2010.

[2] W. F. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz, "Experimental evaluation in computer science: A quantitative study," *J. Syst. Softw.*, vol. 28, no. 1, pp. 9–18, 1995.

[3] D. G. Feitelson, "Experimental computer science: The need for a cultural change," 2006. [Online]. Available: http://www.cs.huji.ac.il/~feit/papers/exp05.pdf, Accessed: Feb. 2019.

[4] J. Krüger and R. Westermann, "Acceleration techniques for GPU-based volume rendering," in *Proc. 14th IEEE Vis.*, 2003, Art. no. 38.

[5] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl, "A simple and flexible volume rendering framework for graphics-hardware–based raycasting," in *Proc. Vol. Graph.*, 2005, pp. 187–195.

[6] M. Hadwiger, P. Ljung, C. R. Salama, and T. Ropinski, "Advanced illumination techniques for GPU-based volume raycasting," in *Proc. SIGGRAPH Courses*, 2009, pp. 2:1–2:166.

[7] J. Beyer, M. Hadwiger, and H. Pfister, "State-of-the-art in GPU-based large-scale volume visualization," *Comput. Graph. Forum*, vol. 34, no. 8, pp. 13–37, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12605

[8] E. W. Bethel and M. Howison, "Multi-core and many-core shared-memory parallel raycasting volume rendering optimization and tuning," *Int. J. High Perform. Comput. Appl.*, vol. 26, no. 4, pp. 399–412, Nov. 2012.

[9] C. Bentes, B. B. Labronici, L. M. Drummond, and R. Farias, "Towards an efficient parallel raycasting of unstructured volumetric data on distributed environments," *Cluster Comput.*, vol. 17, no. 2, pp. 423–439, Jun. 2014.

[10] M. Larsen, S. Labasan, P. A. Navrátil, J. S. Meredith, and H. Childs, "Volume rendering via data-parallel primitives," in *Proc. Eurographics Symp. Parallel Graph. Vis.*, 2015, pp. 53–62.

[11] J. E. Stone, J. Saam, D. J. Hardy, K. L. Vandivort, W.-M. W. Hwu, and K. Schulten, "High performance computation and interactive display of molecular orbitals on GPUs and multi-core CPUs," in *Proc. 2nd Workshop General Purpose Process. Graph. Process. Units*, 2009, pp. 9–18.

[12] P. Kipfer, M. Segal, and R. Westermann, "Uberflow: A GPU-based particle engine," in *Proc. Graph. Hardware*, 2004, pp. 115–122.

[13] T. Harada, S. Koshizuka, and Y. Kawaguchi, "Smoothed particle hydrodynamics on GPUs," in *Proc. Comput. Graph. Int.*, 2007, vol. 40, pp. 63–70.

[14] S. Gumhold, "Splatting illuminated ellipsoids with depth correction," in *Proc. Symp. Vis. Model. Vis.*, 2003, pp. 245–252.

[15] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 152–163, Jun. 2009.

[16] T. Deakin, J. Price, M. Martineau, and S. McIntosh-Smith, "S. GPU-stream v2.0: Benchmarking the achievable memory bandwidth of many-core processors across diverse parallel programming models," in *Proc. Int. Conf. High Perform. Comput.*, 2016, pp. 489–507.

[17] E. Konstantinidis and Y. Cotronis, "A practical performance model for compute and memory bound GPU kernels," in *Proc. Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process.*, 2015, pp. 651–658.

[18] Y. Zhang and J. D. Owens, "A quantitative performance analysis model for GPU architectures," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, 2011, pp. 382–393.

[19] S. Rizzi, M. Hereld, J. A. Insley, M. E. Papka, T. D. Uram, and V. Vishwanath, "Performance modeling of vl3 volume rendering on GPU-based clusters," in *Proc. Eurographics Symp. Parallel Graph. Vis.*, 2014, pp. 65–72.

[20] G. Tkachev, S. Frey, C. Müller, V. Bruder, and T. Ertl, "Prediction of distributed volume visualization performance to support render hardware acquisition," in *Proc. Eurographics Symp. Parallel Graph. Vis.*, 2017, pp. 11–20.

[21] M. Larsen, C. Harrison, J. Kress, D. Pugmire, J. S. Meredith, and H. Childs, "Performance modeling of in situ rendering," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2016, pp. 276–287.

[22] M. Wimmer and P. Wonka, "Rendering time estimation for real-time rendering," in *Proc. Eurographics Workshop Rendering*, 2003, pp. 118–129.

[23] N. Tack, F. Morán, G. Lafruit, and R. Lauwereins, "3D graphics rendering time modeling and control for mobile terminals," in *Proc. 9th Int. Conf. 3D Web Technol.*, 2004, pp. 109–117.

[24] I. Bowman, J. Shalf, K.-L. Ma, and W. Bethel, "Performance modeling for 3D visualization in a heterogeneous computing environment," Lawrence Berkeley National Laboratory, Berkeley, CA, USA, Tech. Rep. LBNL-56977, 2004.

[25] M. Hadwiger, A. K. Al-Awami, J. Beyer, M. Agus, and H. Pfister, "Sparseleap: Efficient empty space skipping for large-scale volume rendering," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 974–983, Aug. 2018.

[26] J. G. Magnus and S. Bruckner, "Interactive dynamic volume illumination with refraction and caustics," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 984–993, Jan. 2018.

[27] J. Wang, F. Yang, and Y. Cao, "A cache-friendly sampling strategy for texture-based volume rendering on GPU," *Visual Informat.*, vol. 1, no. 2, pp. 92–105, 2017.

[28] D. Jönsson and A. Ynnerman, "Correlated photon mapping for interactive global illumination of time-varying volumetric data," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 901–910, Jan. 2017.

[29] K. Wu, A. Knoll, B. J. Isaac, H. Carr, and V. Pascucci, "Direct multi-field volume ray casting of fiber surfaces," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 941–949, Jan. 2017.

[30] F. Sans and R. Carmona, "Volume ray casting using different GPU based parallel apis," in *Proc. XLII Latin Amer. Comput. Conf.*, Oct. 2016, pp. 1–11.

[31] Y. Zhang and K.-L. Ma, "Decoupled shading for real-time heterogeneous volume illumination," *Comput. Graph. Forum*, vol. 35, no. 3, pp. 401–410, 2016.

[32] M. Ament and C. Dachsbacher, "Anisotropic ambient volume shading," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 1015–1024, Jan. 2016.
[33] V. Bruder, S. Frey, and T. Ertl, "Real-time performance prediction and tuning for interactive volume raycasting," in *Proc. SIGGRAPH ASIA Symp. Vis.*, 2016, pp. 7:1–7:8.
[34] T. Zhang, Z. Yi, J. Zheng, D. C. Liu, W.-M. Pang, Q. Wang, and J. Qin, "A clustering-based automatic transfer function design for volume visualization," *Math. Problems Eng.*, vol. 2016, 2016, Art. no. 4547138.
[35] Z.-Y. Ding, J.-G. Tan, X.-Y. Wu, W.-F. Chen, F.-R. Wu, X. Li, and W. Chen, "A near lossless compression domain volume rendering algorithm for floating-point time-varying volume data," *J. Vis.*, vol. 18, no. 2, pp. 147–157, 2015.
[36] Y. Sugimoto, F. Ino, and K. Hagihara, "Improving cache locality for GPU-based volume rendering," *Parallel Comput.*, vol. 40, no. 5/6, pp. 59–69, 2014.
[37] R. Hero, C. Ho, and K.-L. Ma, "Volume rendering of curvilinear-grid data using low-dimensional deformation textures," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 9, pp. 1330–1343, Sep. 2014.
[38] S. Frey, F. Sadlo, K.-L. Ma, and T. Ertl, "Interactive progressive visualization with space-time error control," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 2397–2406, Dec. 2014.
[39] B. Lee and Y.-G. Shin, "Advanced interactive preintegrated volume rendering with a power series," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 8, pp. 1264–1273, Aug. 2013.
[40] B. Liu, G. J. Clapworthy, F. Dong, and E. C. Prakash, "Octree rasterization: Accelerating high-quality out-of-core GPU volume rendering," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 10, pp. 1732–1745, Oct. 2013.
[41] F. Yang, Q. Li, D. Xiang, Y. Cao, and J. Tian, "A versatile optical model for hybrid rendering of volume data," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 6, pp. 925–937, Jun. 2012.
[42] D. Jönsson, J. Kronander, T. Ropinski, and A. Ynnerman, "Historygrams: Enabling interactive global illumination in direct volume rendering using photon mapping," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2364–2371, Dec. 2012.
[43] J. Kronander, D. Jonsson, J. Low, P. Ljung, A. Ynnerman, and J. Unger, "Efficient visibility encoding for dynamic illumination in direct volume rendering," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 3, pp. 447–462, Mar. 2012.
[44] P. Schlegel, M. Makhinya, and R. Pajarola, "Extinction-based shading and illumination in GPU volume ray-casting," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 1795–1802, Dec. 2011.
[45] F. Hernell, P. Ljung, and A. Ynnerman, "Local ambient occlusion in direct volume rendering," *IEEE Trans. Vis. Comput. Graph.*, vol. 16, no. 4, pp. 548–559, Jul./Aug. 2010.
[46] J. Zhou and M. Takatsuka, "Automatic transfer function generation using contour tree controlled residue flow model and color harmonics," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 6, pp. 1481–1488, Nov./Dec. 2009.
[47] C. Lundström, P. Ljung, A. Persson, and A. Ynnerman, "Uncertainty visualization in medical volume rendering using probabilistic animation," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 6, pp. 1648–1655, Dec. 2007.
[48] P. Ljung, C. Winskog, A. Persson, C. Lundstrom, and A. Ynnerman, "Full body virtual autopsies using a state-of-the-art volume rendering pipeline," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 869–876, Sep./Oct. 2006.
[49] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl, "A simple and flexible volume rendering framework for graphics-hardware-based raycasting," in *Proc. Vol. Graph.*, 2005, pp. 187–241.
[50] S. Bruckner and M. E. Gröller, "Volumeshop: An interactive system for direct volume illustration," in *Proc. IEEE Vis.*, 2005, pp. 671–678.
[51] J. H. Krüger and R. Westermann, "Acceleration techniques for GPU-based volume rendering," in *Proc. IEEE Vis.*, 2003, pp. 287–292.
[52] M. Hadwiger, F. Laura, C. Rezk-Salama, T. Höllt, G. Geier, and T. Pabel, "Interactive volume exploration for feature detection and quantification in industrial ct data," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 6, pp. 1507–1514, Nov./Dec. 2008.
[53] C. Müller, M. Krone, M. Huber, V. Biener, D. Herr, S. Koch, G. Reina, D. Weiskopf, and T. Ertl, "Interactive molecular graphics for augmented reality using hololens," *J. Integrative Bioinformatics*, vol. 15, pp. 1–13, 2018.
[54] M. Ibrahim, P. Wickenhäuser, P. Rautek, G. Reina, and M. Hadwiger, "Screen-space normal distribution function caching for consistent multi-resolution rendering of large particle data," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 944–953, Jan. 2018.
[55] P. Hermosilla, M. Krone, V. Guallar, P.-P. Vázquez, A. Vinacua, and T. Ropinski, "Interactive GPU-based generation of solvent-excluded surfaces," *Vis. Comput.*, vol. 33, no. 6–8, pp. 869–881, 2017.
[56] A. Jurčík, J. Parulek, J. Sochor, and B. Kozlikova, "Accelerated visualization of transparent molecular surfaces in molecular dynamics," in *Proc. PacificVis*, 2016, pp. 112–119.
[57] R. Skånberg, P.-P. Vázquez, V. Guallar, and T. Ropinski, "Real-time molecular visualization supporting diffuse interreflections and ambient occlusion," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 718–727, Jan. 2016.
[58] S. Grottel, M. Krone, C. Müller, G. Reina, and T. Ertl, "Megamol – a prototyping framework for particle-based visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 21, no. 2, pp. 201–214, Feb. 2015.
[59] I. Wald, A. Knoll, G. P. Johnson, W. Usher, V. Pascucci, and M. E. Papka, "CPU ray tracing large particle data with balanced pkd trees," in *Proc. Sci. Vis. Conf.*, 2015, pp. 57–64.
[60] D. Guo, J. Nie, M. Liang, Y. Wang, Y. Wang, and Z. Hu, "View-dependent level-of-detail abstraction for interactive atomistic visualization of biological structures," *Comput. Graph.*, vol. 52, pp. 62–71, 2015.
[61] A. Knoll, I. Wald, P. Navratil, A. Bowen, K. Reda, M. E. Papka, and K. Gaither, "RBF volume ray casting on multicore and many-core CPUs," *Comput. Graph. Forum*, vol. 33, no. 3, pp. 71–80, 2014.
[62] M. Le Muzic, J. Parulek, A.-K. Stavrum, and I. Viola, "Illustrative visualization of molecular reactions using omniscient intelligence and passive agents," *Comput. Graph. Forum*, vol. 33, pp. 141–150, 2014.
[63] S. Grottel, M. Krone, K. Scharnowski, and T. Ertl, "Object-space ambient occlusion for molecular dynamics," in *Proc. PacificVis*, 2012, pp. 209–216.
[64] N. Lindow, D. Baum, and H.-C. Hege, "Interactive rendering of materials and biological structures on atomic and nanoscopic scale," *Comput. Graph. Forum*, vol. 31, pp. 1325–1334, 2012.
[65] M. Chavent, A. Vanel, A. Tek, B. Levy, S. Robert, B. Raffin, and M. Baaden, "GPU-accelerated atom and dynamic bond visualization using hyperballs: A unified algorithm for balls, sticks, and hyperboloids," *J. Comput. Chem.*, vol. 32, no. 13, pp. 2924–2935, 2011.
[66] N. Lindow, D. Baum, S. Prohaska, and H.-C. Hege, "Accelerated visualization of dynamic molecular surfaces," *Comput. Graph. Forum*, vol. 29, pp. 943–952, 2010.
[67] S. Grottel, G. Reina, C. Dachsbacher, and T. Ertl, "Coherent culling and shading for large molecular dynamics visualization," *Comput. Graph. Forum*, vol. 29, no. 3, pp. 953–962, 2010.
[68] M. Krone, K. Bidmon, and T. Ertl, "Interactive visualization of molecular surface dynamics," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 6, pp. 1391–1398, Nov./Dec. 2009.
[69] M. Falk, M. Klann, M. Reuss, and T. Ertl, "Visualization of signal transduction processes in the crowded environment of the cell," in *Proc. PacificVis*, 2009, pp. 169–176.
[70] O. D. Lampe, I. Viola, N. Reuter, and H. Hauser, "Two-level approach to efficient visualization of protein dynamics," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 6, pp. 1616–1623, Nov./Dec. 2007.
[71] C. P. Gribble, T. Ize, A. Kensler, I. Wald, and S. G. Parker, "A coherent grid traversal approach to visualizing particle-based simulation data," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 4, pp. 758–768, Jul./Aug. 2007.
[72] M. Tarini, P. Cignoni, and C. Montani, "Ambient occlusion and edge cueing for enhancing real time molecular visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 1237–1244, Sep./Oct. 2006.
[73] G. Reina and T. Ertl, "Hardware-accelerated glyphs for mono- and dipoles in molecular dynamics visualization," in *Proc. EuroVis*, 2005, pp. 177–182.
[74] T. Klein and T. Ertl, "Illustrating magnetic field lines using a discrete particle model," in *Proc. Symp. Vis. Model. Vis.*, vol. 4, 2004, pp. 387–394.
[75] C. Schulz, A. Nocaj, M. El-Assady, S. Frey, M. Hlawatsch, M. Hund, G. Karch, R. Netzel, C. Schätzle, M. Butt, et al., "Generative data models for validation and evaluation of visualization techniques," in *Proc. 6th Workshop Beyond Time Errors Novel Eval. Methods Vis.*, 2016, pp. 112–124.

Valentin Bruder received the master's degree in computer science from the University of Stuttgart, Germany. He is working toward the PhD degree at the University of Stuttgart Visualization Research Center (VISUS). His research interests include new methods to assess, model, and predict performance of visual computing systems with a focus on scientific visualization algorithms.

Christoph Müller received the diploma in computer science from the University of Stuttgart, Germany. He is working toward the PhD degree at the Visualization Research Center of the University of Stuttgart (VISUS). His research interests include graphics clusters and tiled displays.

Steffen Frey received the PhD degree in computer science from the University of Stuttgart, Germany. Currently, he is a postdoc with the University of Stuttgart Visualization Research Center (VISUS). His research interests include visual analysis techniques for large and complex data in scientific visualization, with a particular focus on performance-related aspects and expressive visual representations of dynamic processes.

Thomas Ertl received the MS degree in computer science from the University of Colorado at Boulder and the PhD degree in theoretical astrophysics from the University of Tuebingen. He is a full professor of computer science with the University of Stuttgart, Germany, and the head with the Visualization and Interactive Systems Institute (VIS) and with the Visualization Research Center (VISUS). His research interests include visualization, computer graphics, and human computer interaction. He is a life fellow of the IEEE.