Vision, Modeling, and Visualization (2022)
J. Bender, M. Botsch, and D. Keim (Eds.)

Visually Comparing Rendering Performance from Multiple Perspectives

Hagen Tarner[1], Valentin Bruder[2], Steffen Frey[3], Thomas Ertl[2], and Fabian Beck[4]

[1] University of Duisburg-Essen, Germany
[2] University of Stuttgart, Germany
[3] University of Groningen, The Netherlands
[4] University of Bamberg, Germany

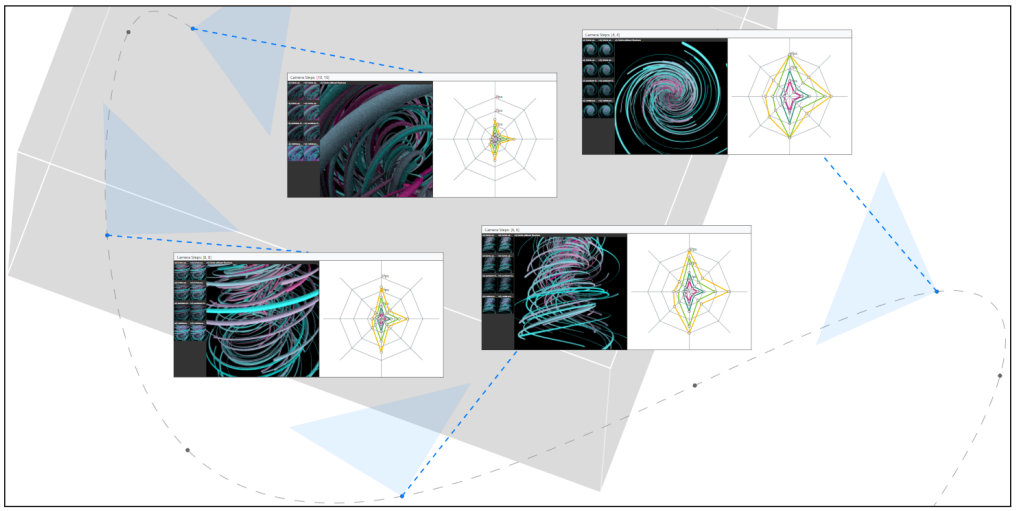

Figure 1: *Performance comparison of different streamline rendering techniques at different views—represented via blue nodes with stylized frustums—along a camera path (stippled gray line). The data domain is depicted by a gray bounding box for context. Performance data is conveyed via radar charts with one axis per technique, accompanied by corresponding rendered images. This clearly shows the different performance characteristics across techniques and how they vary under changing camera configurations.*

Abstract

Evaluation of rendering performance is crucial when selecting or developing algorithms, but challenging as performance can largely differ across a set of selected scenarios. Despite this, performance metrics are often reported and compared in a highly aggregated way. In this paper we suggest a more fine-grained approach for the evaluation of rendering performance, taking into account multiple perspectives on the scenario: camera position and orientation along different paths, rendering algorithms, image resolution, and hardware. The approach comprises a visual analysis system that shows and contrasts the data from these perspectives. The users can explore combinations of perspectives and gain insight into the performance characteristics of several rendering algorithms. A stylized representation of the camera path provides a base layout for arranging the multivariate performance data as radar charts, each comparing the same set of rendering algorithms while linking the performance data with the rendered images. To showcase our approach, we analyze two types of scientific visualization benchmarks.

1. Introduction

Evaluating the performance of graphics and visualization rendering techniques is essential when introducing a new approach or choosing a suitable technique for a given scenario. This generally requires analysis and comparison of metrics such as frame rates. Benchmarks are generally designed to cover representative

runs and are used to measure the overall performance of a technique. Comprehensive benchmarks generate numerous measurements which generally vary significantly depending on a wide range of influencing factors like camera configuration, compute hardware, resolution, etc. Detailed analysis of the characteristics of this complex parameter space is a challenging problem, and for practical purposes the measurements are often condensed into one or few scalar numbers (e.g., average frame rate or percentiles). However, aggregating performance to a single or few values provides only a limited, superficial perspective as performance differences between rendering techniques might interact strongly with the above influence factors. Better understanding these detailed differences could help, for instance, selecting the best technique for a specific scene, avoiding problematic camera perspectives, informing a fair comparative evaluation, detecting quality mismatches between the techniques, and improving a technique.

Our approach, hence, is more fine-grained and supports the visual comparison of rendering performance taking into account multiple perspectives: different camera paths, different rendering techniques, and different hardware. Following the movement of the camera through a scene and using this as a basis for visualization provides a natural contextualization of the data and eases interpretation—connecting the performance data to the camera view is key for understanding view-specific performance characteristics. We specifically support the following analysis tasks:

- **T1:** Compare the performance of rendering techniques for a specific camera configuration in context of the rendered images.
- **T2:** Show the performance of rendering techniques as part of a camera path and support the comparison of different paths.
- **T3:** Identify clusters and respective outliers of data points with similar performance characteristics across the techniques.
- **T4:** Study interactions of performance with other rendering parameters and hardware setups.

In the visualization, as shown in Figure 1, techniques are compared with respect to different camera configurations, and performance data is linked with rendered images. A stylized representation of the camera path and the bounding box of the main scene (gray area) provide the base layout. Multivariate performance radar charts compare the same set of rendering techniques next to the respective renderings (**T1** and **T2**). Integrating this visualization as a main view, we developed a visual analysis system for comparing multivariate rendering performance. Additional views compare the performance of different rendering techniques in scatterplots and tables (see Figure 3; **T3**). Clusters and outliers can be selected for closer analysis and connected to specific setups (**T4**). This is complemented by an overview of all recorded camera paths (see Figure 2; **T1** and **T2**). The supplemental material contains a video demonstrating interactions and a typical analysis workflow.

With this approach and system, which we call *Multiple Perspectives*, we target scientific visualization developers and researchers. Our work supports them in evaluating rendering techniques as well as forming hypotheses for improving the techniques, which we showcase in two application examples (see Section 5 and the supplemental video). We have developed the approach in close collaboration of researchers from the area of performance and software visualization and of scientific visualization.

2. Related Work

The proposed approach both relates to scientific visualization as part of the problem domain (i.e., rendering performance) as well as information visualization and visual analytics as part of the solution domain (i.e., visual analysis of performance data).

Evaluating Rendering Performance Several works on evaluating rendering performance of scientific visualizations conclude that performance substantially varies based on the rendered dataset, hardware, and parameters such as the camera perspective, often in non-obvious ways [BH12; LHK*16; BMFE19; BLE*22]. Benchmarks such as SPECviewperf [Sta21] have been proposed to generate a standard for measuring graphics performance. However, typically, performance of scientific visualization algorithms is still reported only as runtime speed in a single figure or table, evaluated on a single hardware platform for a few different datasets [IIC*13; BMFE19]. Some works, in contrast, contain a more fine-grained evaluation and present results on runtime behavior for different camera paths, dataset sizes, or differences in image quality. Wang et al. [WYC17], for instance, conduct an in-depth performance evaluation in their work on a cache-friendly sampling strategy for volume rendering. They consider multiple viewports, datasets, and camera rotations, while only evaluating one GPU. Other volume rendering works use hundreds of different camera views by recording user interaction sequences for several datasets, but limit their evaluation to one viewport and device [FSME14; BFE17]. Regarding particle visualization, for example, Wald et al. [WKJ*15] and Knoll et al. [KWN*14] evaluate several devices, datasets, and two views each. But despite these individual examples, overall, performance evaluation in scientific visualization will profit if researchers and developers can visually explore the performance data.

Performance Visualization In both industry and academia, visualization is an integral part of the performance analysis of rendering algorithms. There are various vendor- or hardware-specific tools that visually assist in profiling, finding bottlenecks, and optimizing performance [AMD21; NVI21; GGA11]. They typically allow for a detailed analysis of (low-level) hardware specifics, but do not embed the results within the context of the rendered scene and do not explicitly support the comparison of different techniques. Not specific to rendering, Isaacs et al. [IGJ*14] review numerous general performance visualization techniques, and also describe application-specific solutions. For instance, performance can be visually mapped to the physical space of a simulation. Following this line, Schulz et al. [SLB*11] observe that application developers often find visualized application contexts to be highly intuitive when analyzing runtime performance. They uncovered performance bottlenecks by visually arranging performance counters of processors by the physical region they simulated. Similarly, Wylie and Geimer [WG11] reveal performance bottlenecks by attributing the physical space with computation times. With our technique, we do not visualize simulation performance but provide application context, here camera perspectives. Moreover, our approach also relates to visual comparison [GAW*11; LJS21] of performance data, especially if performance is described as multivariate data [SH94; PDW*14]. To visualize it, researchers use standard techniques for multivariate data visualization [TFPB18; TFPB20] or specifically tailored visualizations [SBB19; BBRB12; PGFL05]. In our work,

we use a highly interactive custom visualization to compare the performance of different rendering techniques and their outputs.

3. Rendering Performance Data

Rendering performance can be measured from multiple perspectives, and it is important to consider these within an evaluation. Scene $s \in S$ and camera configuration $c_k \in C_k$ define the image content, and resolution $r \in R$ specifies the number of pixels. Camera configurations c_k belong to a sequential camera path $C_k \in C$. With this, a rendering algorithm and corresponding parameter setting $p \in P$ are employed; the combination of algorithm and respective parameters defines a specific instantiation that is referred to as *technique* throughout this paper. A technique is executed on a certain hardware architecture and model $h \in H$. In total, the synthesis γ of an image $i \in I$ yields an execution performance $x \in X$:

$$(x, i) = \gamma_{s,C_k,p,h,r}(c_k). \tag{1}$$

The camera configuration c_k plays an important role as it is constantly varied in interactive approaches, while the other impact factors typically remain constant. Camera paths C_k naturally reflect a specific animated scene navigation of a user. Also, the ordering can have a performance impact through caching effects—it might matter in which order the respective configurations were rendered, and rendering gets more efficient if previous rendering results can be partly reused. The scene s and camera configuration c_k in particular have an impact on the image itself, but the technique p generally also influences the quality. Depending on the parameter, available options P might be numeric (e.g., step size along a ray in volume raycasting), categorical (e.g., graphics cards $\in H$), or combinations thereof. While any performance metric can be represented by X, we consistently use *frames per second* (fps), as a common metric, and have optimized the visualizations for its properties.

For a (comparative) analysis of each variable's impact, we systematically sample the input parameter space (S, C, P, R, H) of Equation 1. In practice, a problem is though that testing many parameters and configurations of each influential factor is difficult as the number of necessary runs quickly 'explodes' ($|S| \cdot \sum_k |C_k| \cdot |P| \cdot |R| \cdot |H|$). Hence, parameters need to be restricted to a small number of essential ones and only varied across few configurations. We, hence, focus on the rendering part of the visualization pipeline, not considering filtering and mapping operations that generate the renderable representation (scene s) from raw data. One specific sample yields a vector (s, c_k, p, r, h, x, i), with $(x, i) = \gamma_{s,C_k,p,h,r}(c_k)$ (see Equation 1). Since we focus on comparing techniques $p \in P$, as a basis for these investigations, a *data point* constitutes a vector of performance measurements $(x_p \forall p \in P)$ with $(x_p, i_p) \leftarrow \gamma_{\dot{s},\dot{C}_k,p,\dot{h},\dot{r}}(\dot{c}_k)$ for fixed $\dot{s} \in S, \dot{c}_k \in \dot{C}_k, \dot{r} \in R$, and $\dot{h} \in H$.

4. Multiple Perspectives – Visualization Approach

We propose *Multiple Perspectives*, a visual analysis approach for analyzing rendering performance. The design of the system was refined in short iterations, discussing each intermediate stage within the team of authors, which comprised two experts in software performance visualization and two domain experts in scientific visualization. Mostly regular (bi-)weekly meetings for about one year have driven the development of the approach and provided continuous feedback. This process has led from first prototypes that consisted only of standard multivariate data visualizations (e.g., scatterplots, tables, parallel coordinates plots) to the current highly tailored approach. With each iteration of prototypes, we also refined the list of analysis tasks, which in turn reflected back on the prototype and guided central design decisions.

During this process, we identified the following three key design decisions: (I) use radar charts as the main way of depicting multivariate performance data, (II) split the input parameter space into meaningful subsets for visualization based on different scenes and camera paths, and (III) use a projected camera path as a base layout for the main visualization. Radar charts can be considered as a variant of parallel coordinates plots where the axes are arranged in a radial layout. In contrast to parallel coordinates plots, they usually only show one or few data items, but the lines form characteristic shapes that can act as fingerprints or indicators of visual similarity. In our case, each axis represents one technique p and shows the median performance measurements for the respective technique. The design decision to use radar charts consistently across all views was motivated by their compact representation, their ability to form interpretable and memorable visual patterns, and their relative simplicity. This decision was made after first using a single parallel coordinate plot to represent the whole dataset of multivariate performance data. Due to visual clutter and scalability issues, we decided to split the data per camera path and sampling point.

We implemented our approach as a web-based system using `Vue.js`, `D3.js`, and `three.js`. The application is divided into the *Dataset Explorer* (Section 4.1) and the *Camera Path Explorer* (Section 4.2). The Dataset Explorer gives an overview over all available camera paths for a selected dataset and their performance characteristics. Selecting one opens the Camera Path Explorer, which features a highly interactive custom visualization to explore rendering performance of a single camera path.

4.1. The Dataset Explorer

After selecting a dataset $s \in S$, the Dataset Explorer shows a grid of small multiples representing all available camera paths $C_k \in C$, each as a combination of a 3D thumbnail and a radar chart (see Figure 2). With this view, we address tasks **T1** and **T2** from a higher level of abstraction, providing a summarized preview of the performance data comparing different camera paths.

On the left of each box that represents a camera path, a zoomable and rotatable 3D thumbnail provides spatial context. It shows the bounding box of the main scene drawn as a gray cuboid. Each camera position c_k along a camera path C_k is rendered as a blue sphere, and the points are connected to indicate the camera movement. Below the 3D representation, we show a set of thumbnails, displaying the rendered images along the camera path. We show only one image i for each sampled camera position c_k. The parameters p, hardware h, and resolution r are set to default values as they typically do not have a noticeable effect on thumbnail-sized images.

To give an overview of the recorded performance metric for each camera path, we use radar charts as motivated above. The data displayed in the chart is filtered by the selected scene s and camera

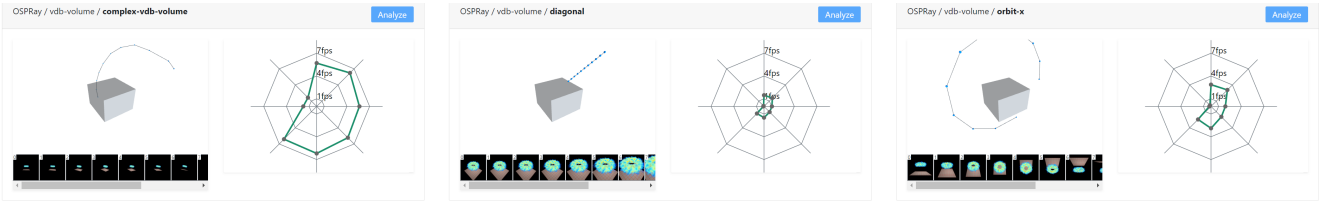

Figure 2: *One row in the Dataset Explorer (Section 4.1) showing for each camera path an interactive 3D abstraction of the path, rendered sample images, and aggregated performance metrics. Depicted are three of the nine recorded paths of the OSPRay example (Section 5.1).*

path C_k, but aggregated across all camera configurations along the path $c_k \in C_k$, resolutions R, and hardware setups H. In other words, the chart summarizes all data points x_p (which are vectors of performance measurements that relate to the camera path, see Section 3) comparing the techniques in P on the different radial axes. In order to better assess the performance values, we add common fps rates (30, 60, 144 fps) as reference lines.

The small multiples of the Dataset Explorer allow for a quick comparison of the characteristics of the different camera paths. With its focus on only two freely selectable parameters for comparison (dataset $s \in S$, and camera path C_k), we effectively split the possibly large input parameter space into manageable subsets, which can be analyzed in more depth in the Camera Path Explorer.

4.2. The Camera Path Explorer

The Camera Path Explorer (Figure 3) addresses all analysis tasks **T1**–**T4** on a detailed level and consists of five views:

I. Camera Path View—a 2D abstraction of the camera path enriched with radar charts comparing the performance of rendering techniques next to the respective rendered images (**T1, T2**).
II. Clustering Panel—an interactive scatterplot that shows a 2D projection of all data points (**T3**).
III. Comparison Panel—a correlation matrix to compare all techniques, from which a scatterplot can be selected for a detailed comparison of two techniques (**T3**).
IV. Faceted Browsing Panel—a faceted browsing interface to subselect data points (**T1, T4**).
V. Data Table—a table with selected data points (**T1, T4**).

The Clustering Panel, Comparison Panel, and Facet Browsing Panel support sub-selection of data points via mouse brushing. All visualizations are linked to display the respective sub-selections.

Camera Path View The sampled camera configurations c_k of the selected three-dimensional camera path C_k serve as a base layout for our main visualization and allow to spatially contextualize the performance information (**T2**). We project the camera path to a non-rotatable plane in order to create a recognizable map-like representation of the path. This simplifies the visual representation and navigation. We determine the plane by minimizing the distance to all sampled camera positions of the respective path. The vertices of the bounding box of the scene, which are shown as a gray cuboid, are projected to the plane using the same method.

To handle camera paths with hundreds of sampling points, we allow automatic and adaptive aggregation into path segments. To identify start and end points of these segments, we give each sampling point an importance value. Importance values are calculated based on the cosine distance of performance metrics of adjacent camera positions. A high cosine distance reflects a change in the multivariate performance data and indicates a candidate for segmentation. Ordering the list of points by importance value prioritizes these candidates. We determine a cut-off value for the list of points that limits the number of generated segments, while still preserving important differences. During development we found our approach to work best with a total segment number between 11 and 30, depending on the currently selected data points. We use a dynamic cut-off value by calculating the pair-wise differences of importance values and finding the maximum value in the range between elements 11 and 30. The resulting candidates are used for segmentation, and all data points of a segment get aggregated. The segmentation adapts to filtering and selection and re-generates segments each time the analyst selects other data points. This way, we can ensure that the application always shows enough information to highlight interesting sampling points, while not overloading the user with too much data. The segments (solid gray lines) are plotted on top of the original camera path (dashed gray line) as can be seen in Figure 3 I. We use the median camera position in the segment as a representative of that segment, for which we add a light blue triangle representing the camera's direction and field of view.

Each segment of the camera path is connected to an overlay box showing the performance of the techniques and images corresponding to the median index (**T1**). The image component on the left consists of two parts: a thumbnail-sized preview of the rendering result, and a magnified version of a selected image. Image selection can be done by hovering a thumbnail (sticky selection by clicking), allowing for a quick comparison of the rendering results. To highlight outliers with unexpected high visual deviations, we calculate the structural similarity (SSIM) [WBSS04] between each image pair of a data point. If the minimum of the SSIM values is below a threshold, we display a red marker indicating major visual differences (see Figure 10). A radar chart that visualizes performance on the right uses the same design as the ones in the Dataset Explorer (Section 4.1). Aggregating all data points x_p for this camera segment, it shows the median fps values with one technique per axis. When data points get interactively sub-selected, the radar chart updates accordingly and only aggregates the current selection. To reduce visual clutter in the Camera Path View, we decided to omit labels on the axes of the various displayed radar charts. Instead of repeating them for every chart, we depict the axis labels as a legend in the top right corner of the screen (see Figure 3 above III).

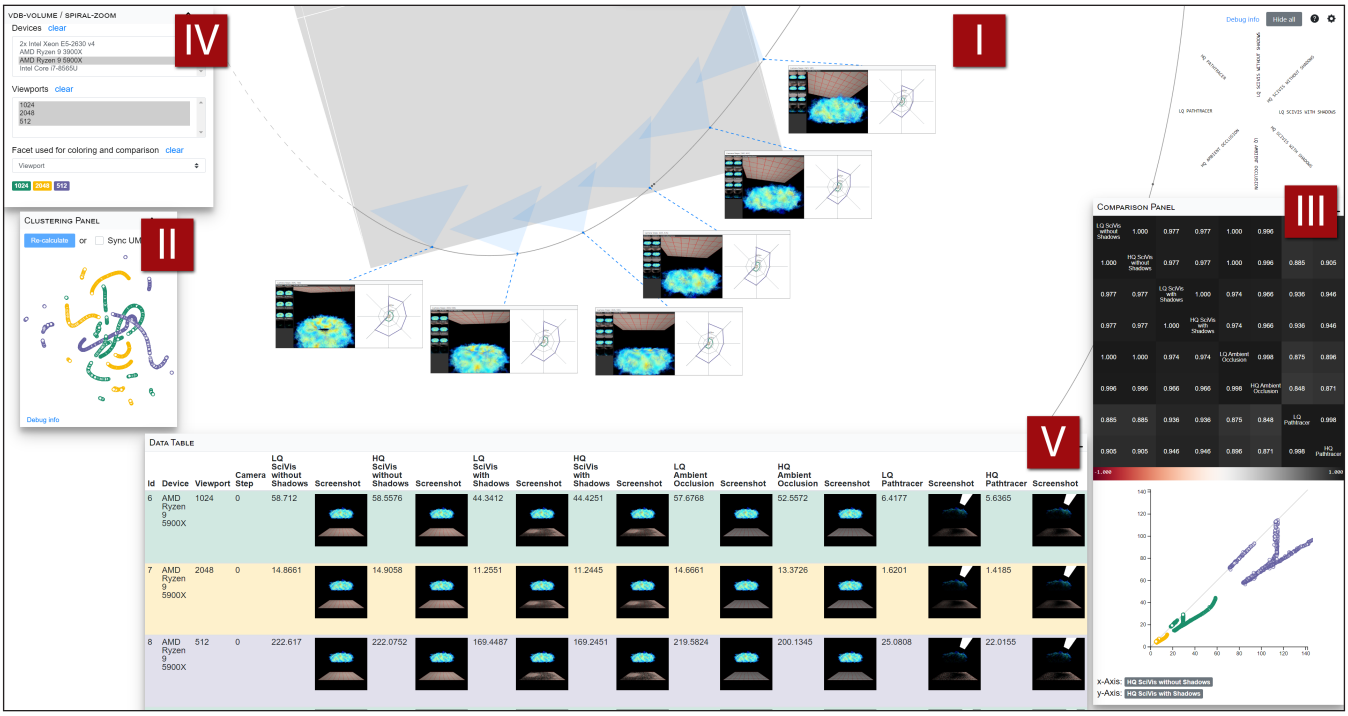

Figure 3: *The Camera Path Explorer (see Section 4.2) comprises (I) a stylized two-dimensional version of the sampled camera path, (II) Clustering Panel, (III) Comparison Panel, (IV) Faceted Browsing Panel, and (V) the Data Table.*

We use a force-directed layout to position the boxes along the path. For this, we attach repelling forces to the boxes, the path sampling points, and the origin point of the 2D projection. Attracting forces are attached to the endpoint pairs of the leaders that connect the boxes with the camera path. This layout method mostly avoids overlapping of boxes with other boxes or path sampling points. But, if necessary, the user can also hide (and show) individual boxes or manually rearrange them via drag and drop. By zooming and panning, the analyst can enlarge specific regions of the projected camera path and overlay boxes to inspect details of the data or images.

Clustering Panel To identify clusters and outliers according (**T3**), the Clustering Panel (Figure 3 II) contains a scatterplot of the data points, where data points of similar performance characteristics across all rendering techniques are plotted in proximity. In particular, we use UMAP [MHM18] to project the multivariate data points, which contain one performance measurement x_p per technique p each (see Section 3), down to two dimensions. The projection is solely based on the performance measurements, and does not take other factors (e.g., camera configurations c_k) into account.

Comparison Panel Complementing the Clustering Panel regarding task **T3**, the Comparison Panel contains a correlation matrix and a scatterplot with interactively selectable axes (Figure 3 III). The correlation matrix shows the Pearson correlation coefficient (*PCC*) for each pairwise combination of techniques. The cells of the matrix are color-coded from red ($PCC = -1$) over white ($PCC = 0$) to black ($PCC = 1$). Selecting a cell of the matrix updates the axes of the scatterplot to visually inspect performance differences between the pair of techniques. In the scatterplot, each point represents one data point x_p. While points close to the diagonal reflect a balanced performance between the two techniques, off-diagonal points indicate a better performance for one of them. It can be discerned, for instance, if better performance relates to a constant factor for all data points or varies for subsets.

Faceted Browsing Panel As discussed in Section 3, we consider various parameters. While scene S, and camera path C_k are selected in the Dataset Explorer, the camera configuration c_k and rendering technique p have dedicated visualizations in the Camera Path View. To also leverage the information about the hardware ($h \in H$) and the image's output resolution ($r \in R$), we introduce additional sub-selection options in the Faceted Browsing Panel (Figure 3 IV). Applying the concept of faceted browsing [YSLH03], we interpret hardware ("Device") and resolution ("Viewports") as facets. Selecting one or more facet values filters the shown data. Selection across different facets is treated as a logical `AND`. Further, a single facet can be selected for further analysis. This will partition the data points along this facet and create an additional line for each facet value in the radar charts of the Camera Path View. Color is used throughout the application to identify the facet values. These interactive selection options allow superimposed comparison of multivariate performance data and address tasks **T1** and **T4**. They allow investigating whether, for instance, different hardware setups influence the performance characteristics along the camera path.

Data Table The Data Table contains all currently selected data points without any aggregation (Figure 3 V). Each table row represents one data point x_p with images i_p as well as the camera configuration c_k ("Camera Step"), the used hardware h ("Device"), and

Table 1: *Devices measured in the application examples: OSPRay used CPUs (top) and the particle rendering GPUs (bottom).*

Vendor	Model	Cores/SUs	Architecture
AMD	Ryzen 9 3900X	12	Zen 2
AMD	Ryzen 9 5900X	12	Zen 3
Intel	2× Xeon E5-2630 v4	2× 10	Broadwell
Intel	Core i7-8565U	4	Whiskey Lake
NVIDIA	Quadro M6000 24GB	3072	Maxwell
NVIDIA	GeForce GTX 1080 Ti	3584	Pascal
NVIDIA	TITAN X (Pascal)	3584	Pascal
NVIDIA	TITAN Xp	3840	Pascal
AMD	Radeon RX 480	2304	GCN 4.0
AMD	Radeon Vega Frontier Edition	4096	GCN 5.0

Table 2: *Techniques used in the OSPRay benchmark. LQ = Low Quality, HQ = High Quality*

Name	Parameter
LQ SciVis with Shadows	ambient occlusion samples: 1
HQ SciVis with Shadows	ambient occlusion samples: 5
LQ SciVis without Shadows	ambient occlusion samples: 1
HQ SciVis without Shadows	ambient occlusion samples: 5
LQ Ambient Occlusion	ambient occlusion samples: 1
HQ Ambient Occlusion	ambient occlusion samples: 5
LQ Path Tracer	ray recursion depth for Russian roulette: 5
HQ Path Tracer	ray recursion depth for Russian roulette: 10

the output resolution r of the rendered image ("Viewport"); scene s and camera path C are redundant for all data points in the screen and hence not included. The Data Table contributes to task **T1** and **T4** by contextualizing the performance measurements with the images as well as the rendering parameters and hardware setups.

5. Application Examples

We demonstrate our approach with two application examples from scientific visualization. In Section 5.1, we investigate the performance of rendering techniques in Intel OSPRay [WJA*16], a library for CPU-based ray tracing. For this, we measured the performance of several CPUs by extending the benchmark capabilities of OSPRay v2.4 for systematic sampling along camera paths. In Section 5.2, we analyze particle visualization techniques using publicly available performance data [BMFE19]. The measured devices are summarized in Table 1.

5.1. OSPRay Volume Visualization

Intel OSPRay is an open source rendering library with high industry adoption. It comes with three different renderers that support different features and materials [Int21].

- The **SciVis renderer** is a fast ray tracer that supports multiple light sources, ambient occlusion, and shadows.
- The **Ambient Occlusion renderer** is a simplified variant of the SciVis renderer that does not consider light sources.
- The **Path Tracer** features indirect illumination, soft shadows, and realistic materials.

We evaluated each of the three renderers with two different parameter configurations, which we refer to as high quality (HQ) and low quality (LQ). For the SciVis and Ambient Occlusion renderer, we change the amount of ambient occlusion samples. In case of the Path Tracer, we vary the ray recursion depth for Russian roulette. Also, in case of the SciVis renderer, we measured the two quality variants with shadows enabled and disabled, respectively. Overall, we ran eight different benchmark variants (Table 2) on four CPUs (Table 1) using three different viewport resolutions (512^2, 1024^2, and 2048^2 pixels). We consider the render times of a generated VDB volume dataset included in OSPRay for benchmarking. The torus-shaped density values are generated using Perlin noise, the transfer function maps them to color. For all techniques, a ground plane is placed below the torus and a rectangular light source is positioned above (see Figure 3 V).

We start our analysis by selecting the "spiral zoom" camera path in the Dataset Explorer. The path is comprised of 1000 different camera configurations and follows an inward spiral towards the dataset. Using the Camera Path Explorer, we first filter for the 512^2 px resolution, to be able to better compare the devices' performance. After coloring by device, we investigate the UMAP plot. All measurements from the Intel i7-8565U CPU form three clusters. By nature, the notebook CPU performs slower than the others, also showing a higher variance. Therefore, we deselect it, producing a division of the camera path into five larger sections (see Figure 4). The radar plots show an expected, decreasing performance as the camera zooms in on the dataset along its path. Differences between the techniques are particularly noticeable (the shape of the radar plot changes, i.e., performance decreases distinctively for the techniques) for the last section of the path where the camera views the dataset from below the ground plane.

Next, we compare pairs of rendering techniques in the correlation matrix of the Comparison Panel. Besides an expected low correlation between the path tracing and the other techniques, a comparably low correlation between the SciVis renderer with and without shadows enabled sticks out. The comparison plot shows a similar, clear pattern for all three devices (Figure 5). While the twelve-core AMD Ryzen 9 3900X slightly outperforms the dual socket Intel Xeon E5 with 20 cores (visible by the offset), the AMD Ryzen 9 5900X is clearly the fastest. Although featuring twelve cores as the 3900X, the 5900X demonstrates its superior instructions per cycle performance. We can roughly divide the comparison scatterplot into four regions A, B, C, and D (see Figure 5), that we brush in each case, to investigate their respective locations on the camera path. Region (A) includes camera positions below the ground plane, where shadows are not visible, resulting in a similar performance no matter if shadows are used or not. Region (B) mainly covers view directions almost parallel to the ground plane; as less and less of the plane becomes visible along the path, the performance of the SciVis renderer with shadows enabled gets faster until no shadow is visible. Similar cases can be found in region (C), where the camera is zoomed out and larger parts of the ground plane get visible. Region (D) shows a near linear correlation between the faster configuration without shadows and the one with shadows enabled.

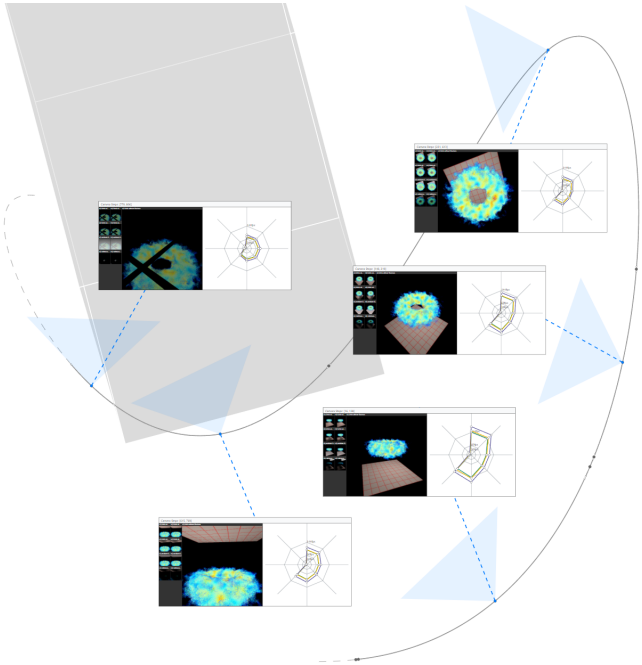

Figure 4: *A spiral path from the OSPRay example. The performance-based segmentation of the path produces five larger stretches.*

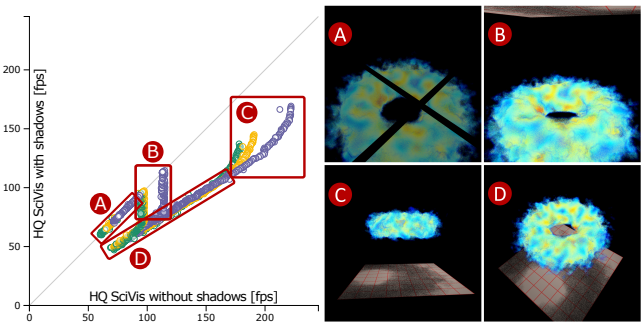

Figure 5: *Comparison of the SciVis renderer in OSPRay with and without shadows enabled. We identified four regions (A–D) with different performance characteristics.*

The radar plots generally indicate a similar performance of the LQ and the HQ variants of the SciVis renderer. This means raising the ambient occlusion parameter seems to have little to no effect on performance. The Comparison Panel confirms our hypothesis. Apparently, the ambient occlusion parameter has no effect on volumes when using the SciVis renderer (with our setup); there are also no noticeable visual differences when comparing the images. However, for the Ambient Occlusion renderer, dark pixels can be seen on the ground plane. We compare the SciVis renderer without using shadows against the Ambient Occlusion renderer, which is supposed to be faster since it does not consider light sources [Int21]. However, the comparison plot shows a better performance of the SciVis renderer across the board. Similar patterns are visible, but not as pronounced: camera configurations below the ground plane

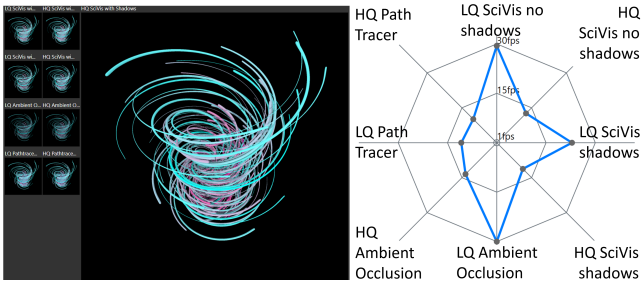

Figure 6: *Techniques show substantial performance differences for a selected case of the streamline dataset.*

perform even better with the SciVis renderer, while the fps-cap for the close-up, side way perspectives (Figure 5B) is also visible.

To check if this is a general issue or limited to volume rendering, we switch to another dataset that contains generated streamlines. The streamline data also comes with OSPRay and consists of 100 randomly generated line geometries, spiraling upwards with changing radius of curvature. Figure 6 shows that, in the case of ray tracing, the opaque streamline objects, the techniques show substantially larger performance deviations compared to the volume dataset. The selected measurements from the Ryzen 9 3900X with a viewport of 512^2 pixels show a clear performance drop when raising the number of ambient occlusion samples for both the SciVis and the Ambient Occlusion renderers. Enabling shadows also shows a noticeable impact, in particular for the LQ variant. With the streamline data, the Ambient Occlusion renderer shows a slightly better performance (around 5%) compared to the SciVis renderer without shadows with HQ/LQ respectively. This aligns with the OSPRay documentation, which attributes the SciVis renderer a slower performance.

The runtimes of the Path Tracer are generally substantially slower than those of the other two renderers (factor 10–15) but hard to compare due to the differences in the approaches, especially for volume datasets. This is also reflected in the rendered images that clearly differ (see Figure 3 V). Selecting a single device and all resolutions, and coloring the plots accordingly, we noticed that performance seems to scale linearly with the viewport resolution throughout both datasets, i.e., quadrupling the pixel count from 512^2 to 1024^2, and again from 1024^2 to 2048^2, typically results in a quarter of the performance.

5.2. Particle Visualization

In the second application example, we examine publicly available data from a recent particle rendering benchmark [BMFE19]. The rendering of such particles, e.g., from molecular data, is typically based on spherical glyphs. The benchmark contains different implementations of this concept, and we perform a close comparison between five techniques. They differ in the shader stages they use to compute the sprites for the glyphs (vertex, geometry, or tessellation shaders) and the form and alignment of the sprites with respect to the camera (ray-aligned or screen-aligned). The benchmark includes the systematic sampling of different parameter configurations, most notably different camera paths. During our investiga-

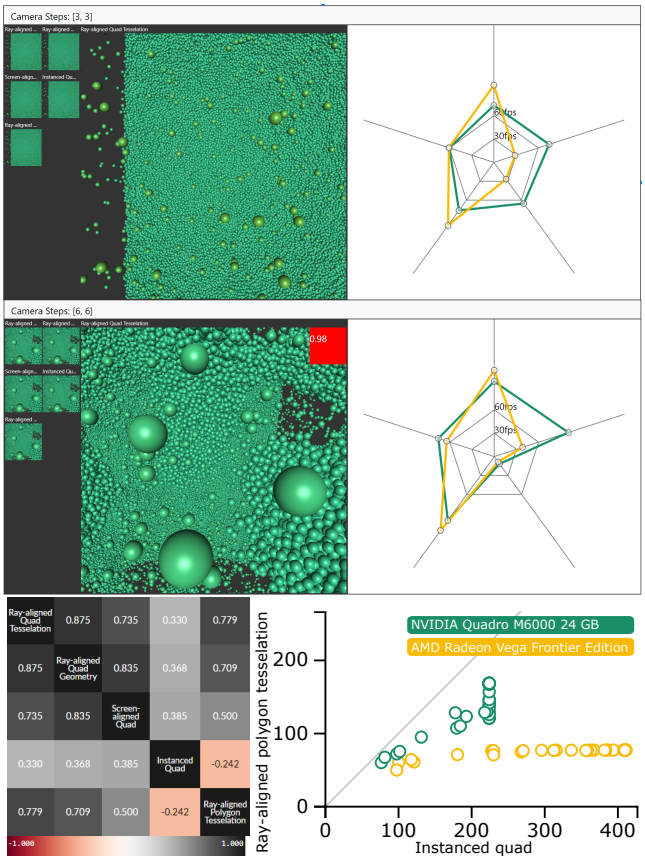

Figure 7: *Comparison of AMD Radeon Vega FE and NVIDIA Quadro M6000. They show different performance characteristics based on rendering technique and camera configuration. Particularly, ray-aligned polygon tessellation and instanced quads differ.*

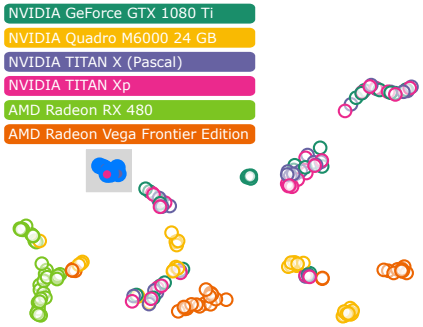

Figure 8: *Selecting a cluster in the Clustering Panel reveals performance similarities for multiple samples. They share the same viewport resolution, GPU architecture and camera positions inside the bounding box with a similar pixel coverage.*

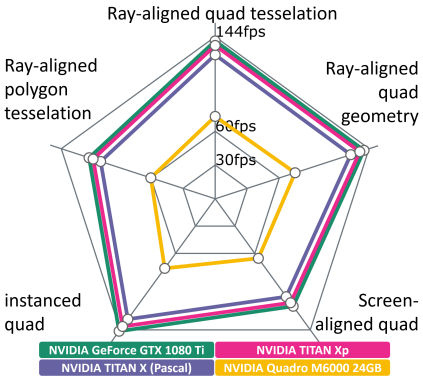

Figure 9: *Performance comparison of all tested NVIDIA cards for a selected camera configuration. The performance characteristics are similar across all devices.*

tion, we take the following parameters into account: dataset, device (i.e., GPUs listed in Table 1), viewport, and the camera setup. The combination of those parameters results in 23 958 configurations. We investigate the dataset of a simulation of a liquid layer formation comprising two million particles.

We start our analysis with the Dataset Explorer, investigating the performance distributions across different camera paths by contrasting the radar plots. While many of the paths show a similar pattern for the five techniques, paths along the y-axis (the longest side of the bounding box) differ. To investigate this more closely, we select the sinusoidal path along the y-axis for further analysis in the Camera Path Explorer. A comparison of the radar charts for all camera positions in the path indicates three regions with different performance characteristics. One covers the first third of the path, where particles are sparsely scattered across the space and the main particle cluster creates a high frequency surface in the distance (e.g., Figure 7). In the second region, the camera moves through the main cluster, resulting in a viewport mostly filled by large spheres (e.g., Figure 10), while in the third part, only few small and scattered particles are rendered. Generally, the frame rates are high, and we filter for the highest viewport resolution in our further analysis.

To investigate similarities between configurations, we check for clusters in the Clustering Panel (Figure 8). Most clusters are groupings of values that either belong to a single GPU, a specific resolution, or combinations thereof. One of the clusters is a group of results from three NVIDIA GPUs with the same Pascal architecture. A closer inspection of the values in the Data Table reveals that the camera positions are located in the middle of the path, which means inside the bounding box. With those configurations, all three graphics cards produce similar fps values.

We then compare the radar charts for all GPUs along the selected camera path further and spot some interesting characteristics in the middle regions of the path. All NVIDIA GPUs show similar performance characteristics—also across architectures (Figure 9). In direct comparison to the NVIDIA Quadro M6000, the Radeon Vega performs substantially better when using the “instanced quads” or the “ray-aligned quad tessellation” methods for rendering (Figure 7). On the other hand, the Quadro M6000 outperforms the AMD GPU when using “screen-aligned quads” or “ray-aligned quad geometry”. Both cards exhibit similar performance when using ray-aligned polygons generated in the tessellation shader.

We notice a similar pattern at the next camera steps. The correlation matrix shows a negative correlation for the ray-aligned poly-

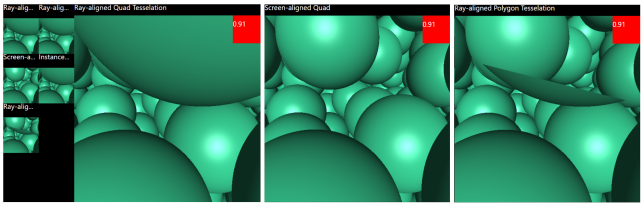

Figure 10: *A red marker indicates a comparably low structural similarity (SSIM) between the images, resulting from different clipping approaches that are influenced by sprite shape and alignment.*

gon tessellation and instanced quads (Figure 7). Selecting the value shows a direct comparison between the two techniques. A horizontal line of values stands out that have similar performance for polygon tessellation, but not for the instanced quads. The coloring by device makes it immediately clear that those belong to the Radeon Vega GPU. The performance on this particular GPU in combination with polygon tessellation seems to be bound at around 80 fps. This effect can be seen across different camera configurations and viewport resolutions. A similar effect, but for the NVIDIA M6000 GPU, can be seen when comparing instanced quads against ray aligned quad tessellation. We further notice a very low performance for the screen-aligned quad technique, hinting at an issue or bottleneck for this configuration. A comparison to other camera paths reveals strong performance drops for the screen-aligned technique whenever the camera gets close to the spherical glyphs.

Comparing the images produced by the various techniques, red indicators imply relatively low structural similarity (SSIM) of at least a subset of the renderings. One of these cases stands (Figure 10) out in particular due to its low SSIM value. When comparing these images, we immediately notice a substantial difference using the screen-aligned quad technique compared to the others. The sprite closest to the camera is (partly) clipped by all techniques except the screen-aligned variant. Different clipping behavior of the techniques may be a cause of performance differences in this configuration. Regarding the performance drop of the screen-aligned technique that can be observed in the renderings for the cases where the camera is inside the bounding box, we suspect a higher overdraw (i.e., overlapping sprites) for close spheres as one of the reasons. Another cause for the performance decrease might be that this method apparently generates sprites for back faces of spheres, causing unnecessary draw calls.

6. Discussion and Future Work

We now discuss current limitations of our approach for visual performance comparison and outline directions for future work.

Scalability As outlined in Section 3, the number of necessary benchmark runs to capture the whole input parameter space explodes with increasing number of impact factors (*curse of dimensionality*). In its current state, the visualization system scales for an increased input parameter space by applying an adaptive sampling algorithm to the camera path, as well as adding new facets and value selectors to the faceted browsing panel. However, the system does not scale well for an increased number of techniques. As each technique is represented by one axis in the radar charts, a larger number might result in cluttered charts. This issue could be mitigated by letting the users select a subset of techniques to analyze.

In Situ Evaluation The current system is used in *a posteriori* scenarios: the camera path, and all performance measurements have to be pre-recorded. A promising direction for future work is an extension to capture performance data while the rendering application is running and display radar charts for different rendering techniques *in situ*. For example, users could interactively select camera positions with uncharacteristic performance for further analysis. Interactive selection of rendering techniques could also be an extension: enabling or disabling additional rendering features, loading different shader configurations, or even manipulating what is shown on screen (number of objects, complexity of geometry, etc.). However, it would be difficult to warrant a systematic sampling in such scenarios, which could limit the comparability of rendering techniques and might require different means of visualization.

Code–Performance Comprehension As discussed in Section 5.2, a direct linking between performance metrics, rendered image, and the source code of the rendering technique helps to understand deviations in rendering output and performance. With an integration of the source code in an additional panel inside the Camera Path Explorer, developers could inspect implementation details, while evaluating corresponding performance and render output. Furthermore, the system could show different versions of the same rendering technique (instead of different techniques), making the evolution of a technique visible while marking code differences. Instead of radar charts, a timeline-based arrangement of the performance metrics would clarify the temporal sequence. These features could answer questions on the performance impact of changes to the source code of the rendering technique and might reveal performance regressions.

Applicability and Extensibility In this paper, we showed the applicability of our approach to datasets from the scientific visualization domain. Due to the close relation of scientific visualization to computer graphics, we assume that our approach also works for this domain. A possible extension would not only include fps values as a metric for comparison, but also other performance metrics such as power or memory consumption. In addition to this, domain-specific new facets could be introduced.

7. Conclusion

We introduced a novel visual analysis approach for analyzing and comparing rendering performance of different techniques on a fine-grained level in the context of the camera paths. It integrates and tailors multivariate data visualizations optimized for visual comparison, collections of the rendered images, projections of the camera paths and scene, and an adaptive sampling algorithm to select camera positions of high importance with regard to performance characteristics. We call the approach *Multiple Perspectives* as the camera paths as well as the other rendering configurations and images provide different perspectives of the rich performance data, which is otherwise aggregated to a level where important details and specifics are obfuscated. In our application examples, we have

demonstrated that the approach helps in typical performance analysis and comparison tasks. For instance, clusters of camera configurations that have special performance characteristics regarding the compared techniques could be revealed, performance bottlenecks for specific device–technique combinations could be uncovered, and non-obvious impact of parameter changes could be identified. In general, we consider our approach as a step towards a more detailed understanding of rendering performance characteristics from (comprehensive) benchmarks, which in turn can contribute to choosing suitable methods and developing more efficient rendering techniques.

8. Acknowledgments

This work has been partly funded by *Deutsche Forschungsgemeinschaft* (DFG) as part of research grant 288909335, as well as within Project A02 of the SFB/Transregio 161 (project number 251654672).

References

[AMD21] AMD. *Radeon GPU Profiler*. https://gpuopen.com/rgp/. accessed March 30, 2021. 2021 2.

[BBRB12] BERGEL, ALEXANDRE, BAÑADOS, FELIPE, ROBBES, ROMAIN, and BINDER, WALTER. "Execution Profiling Blueprints". *Software: Practice and Experience* 42.9 (2012), 1165–1192. DOI: 10.1002/spe.1120 2.

[BFE17] BRUDER, VALENTIN, FREY, STEFFEN, and ERTL, THOMAS. "Prediction-Based Load Balancing and Resolution Tuning for Interactive Volume Raycasting". *Visual Informatics* 1.2 (2017), 106–117. DOI: 10.1016/j.visinf.2017.09.001 2.

[BH12] BETHEL, E WES and HOWISON, MARK. "Multi-core and Many-core Shared-memory Parallel Raycasting Volume Rendering Optimization and Tuning". *The International Journal of High Performance Computing Applications* 26.4 (2012), 399–412. DOI: 10.1177/1094342012440466 2.

[BLE*22] BRUDER, VALENTIN, LARSEN, MATTHEW, ERTL, THOMAS, et al. "A Hybrid In Situ Approach for Cost Efficient Image Database Generation". *IEEE Transactions on Visualization and Computer Graphics* (2022), 1–1. DOI: 10.1109/TVCG.2022.3169590 2.

[BMFE19] BRUDER, VALENTIN, MÜLLER, CHRISTOPH, FREY, STEFFEN, and ERTL, THOMAS. "On Evaluating Runtime Performance of Interactive Visualizations". *IEEE Transactions on Visualization and Computer Graphics* 26.9 (2019). DOI: 10.1109/TVCG.2019.2898435 2, 6, 7.

[FSME14] FREY, STEFFEN, SADLO, FILIP, MA, KWAN-LIU, and ERTL, THOMAS. "Interactive Progressive Visualization with Space-Time Error Control". *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), 2397–2406. DOI: 10.1109/TVCG.2014.2346319 2.

[GAW*11] GLEICHER, MICHAEL, ALBERS, DANIELLE, WALKER, RICK, et al. "Visual Comparison for Information Visualization". *Information Visualization* 10.4 (2011), 289–309. DOI: 10.1177/1473871611416549 2.

[GGA11] GUO, SHENG, GERASIMOV, PHILIPP, and AONA, BONNIE. "Practical Game Performance Analysis Using Intel Graphics Performance Analyzers". *Intel Corporation White Paper* (2011) 2.

[IGJ*14] ISAACS, KATHERINE E, GIMÉNEZ, ALFREDO, JUSUFI, ILIR, et al. "State of the Art of Performance Visualization". *Proceedings of the 16th Eurographics Conference on Visualization*. Eurographics Association, 2014, 141–160. DOI: 10.2312/eurovisstar.20141177 2.

[IIC*13] ISENBERG, TOBIAS, ISENBERG, PETRA, CHEN, JIAN, et al. "A Systematic Review on the Practice of Evaluating Visualization". *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), 2818–2827. DOI: 10.1109/TVCG.2013.126 2.

[Int21] INTEL. *Ospray, the Open, Scalable, and Portable Ray Tracing Engine*. https://www.ospray.org/documentation.html. accessed March 29, 2021. 2021 6, 7.

[KWN*14] KNOLL, AARON, WALD, INGO, NAVRATIL, PAUL, et al. "RBF Volume Ray Casting on Multicore and Manycore CPUs". *Computer Graphics Forum* 33.3 (2014), 71–80. DOI: 10.1111/cgf.12363 2.

[LHK*16] LARSEN, MATTHEW, HARRISON, CYRUS, KRESS, JAMES, et al. "Performance Modeling of In Situ Rendering". *Proceedings of the 28th International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE, 2016, 276–287. DOI: 10.1109/SC.2016.23 2.

[LJS21] L'YI, SEHI, JO, JAEMIN, and SEO, JINWOOK. "Comparative Layouts Revisited: Design Space, Guidelines, and Future Directions". *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2021), 1525–1535. DOI: 10.1109/TVCG.2020.3030419 2.

[MHM18] MCINNES, LELAND, HEALY, JOHN, and MELVILLE, JAMES. "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". *arXiv Preprint* abs/1802.03426 (2018) 5.

[NVI21] NVIDIA. *Visual Profiler*. https://developer.nvidia.com/nsight-visual-studio-edition. accessed March 30, 2021. 2021 2.

[PDW*14] POCO, J., DASGUPTA, A., WEI, Y., et al. "SimilarityExplorer: A Visual Inter-Comparison Tool for Multifaceted Climate Data". *Computer Graphics Forum* 33.3 (2014), 341–350. DOI: 10.1111/cgf.12390 2.

[PGFL05] PINZGER, MARTIN, GALL, HARALD, FISCHER, MICHAEL, and LANZA, MICHELE. "Visualizing Multiple Evolution Metrics". *Proceedings of the 2005 ACM Symposium on Software Visualization*. ACM, 2005, 67–75. DOI: 10.1145/1056018.1056027 2.

[SBB19] SANDOVAL ALCOCER, J. P., BECK, F., and BERGEL, A. "Performance Evolution Matrix: Visualizing Performance Variations along Software Versions". *Proceedings of the 7th IEEE Working Conference on Software Visualization*. 2019, 1–11. DOI: 10.1109/VISSOFT.2019.00009 2.

[SH94] SCHMID, C. and HINTERBERGER, H. "Comparative Multivariate Visualization Across Conceptually Different Graphic Displays". *Proceedings of the 7th International Working Conference on Scientific and Statistical Database Management*. IEEE, 1994, 42–51. DOI: 10.1109/SSDM.1994.336963 2.

[SLB*11] SCHULZ, M., LEVINE, J. A., BREMER, P., et al. "Interpreting Performance Data across Intuitive Domains". *Proceedings of the 40th International Conference on Parallel Processing*. 2011, 206–215. DOI: 10.1109/ICPP.2011.60 2.

[Sta21] STANDARD PERFORMANCE EVALUATION CORPORATION. *SPECviewperf*. https://spec.org/gwpg/gpc.static/vp2020info.html. accessed November 30, 2021. 2021 2.

[TFPB18] TARNER, HAGEN, FRICK, VEIT, PINZGER, MARTIN, and BECK, FABIAN. "Exploring Visual Comparison of Multivariate Runtime Statistics". *Proceedings of the 9th Symposium on Software Performance*. Hildesheim, Germany, 2018 2.

[TFPB20] TARNER, HAGEN, FRICK, VEIT, PINZGER, MARTIN, and BECK, FABIAN. "Visualizing Evolution and Performance Metrics on Method Level As Multivariate Data". *Proceedings of the 13th Seminar Series on Advanced Techniques & Tools for Software Evolution*. CEUR-WS.org, 2020. URL: http://ceur-ws.org/Vol-2754/paper4.pdf 2.

[WBSS04] WANG, Z., BOVIK, A.C., SHEIKH, H.R., and SIMONCELLI, E.P. "Image Quality Assessment: From Error Visibility to Structural Similarity". *IEEE Transactions on Image Processing* 13.4 (2004), 600–612. DOI: 10.1109/TIP.2003.819861 4.

[WG11] WYLIE, BRIAN JN and GEIMER, MARKUS. "Large-scale Performance Analysis of PFLOTRAN with Scalasca". *Proceedings of the 53rd Cray User Group meeting*. Vol. 9. 2011, 12 2.

[WJA*16] WALD, INGO, JOHNSON, GREGORY P, AMSTUTZ, JEFFERSON, et al. "OSPRay – a CPU Ray Tracing Framework for Scientific Visualization". *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2016), 931–940. DOI: 10.1109/tvcg.2016.2599041 6.

[WKJ*15] WALD, INGO, KNOLL, AARON, JOHNSON, GREGORY P., et al. "CPU Ray Tracing Large Particle Data with Balanced P-k-d Trees". *Proceedings of the 2015 IEEE Scientific Visualization Conference*. 2015, 57–64. DOI: 10.1109/SciVis.2015.7429492 2.

[WYC17] WANG, JUNPENG, YANG, FEI, and CAO, YONG. "A Cache-Friendly Sampling Strategy for Texture-Based Volume Rendering on GPU". *Visual Informatics* 1.2 (2017), 92–105. DOI: 10.1016/j.visinf.2017.08.001 2.

[YSLH03] YEE, KA-PING, SWEARINGEN, KIRSTEN, LI, KEVIN, and HEARST, MARTI. "Faceted Metadata for Image Search and Browsing". *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2003, 401–408. DOI: 10.1145/642611.642681 5.