Auto-Tuning Intermediate Representations for In Situ Visualization

Steffen Frey and Thomas Ertl

Abstract—Advances in high-accuracy measurement techniques and parallel computing systems for simulations lead to a widening gap between the rate at which data is generated and the rate at which it can be transferred and stored. In situ visualization directly tackles this issue by processing—and with this reducing—data as soon as it is generated. This allows to create, transmit and store visualizations at a much higher resolution than what would be possible otherwise with traditional approaches. So-called hybrid in situ visualization is a popular variant that transforms data into an intermediate visualization representation of reduced size. These intermediate representations condense the original data by applying visualization techniques, but in contrast to the traditional result of a rendered image, they still preserve some degrees of freedom for live and a posteriori exploration and analysis. However, the configuration of the involved processing steps requires careful configuration under the consideration of achieved quality and preserved degrees of freedom against bandwidth and storage resources.

To optimize the generation of intermediate representations for hybrid in situ visualization, we present our approach to (1) analyze and quantify the impact of input parameters, and (2) to auto-tune them on this basis under the consideration of different constraints. We demonstrate its application and evaluate respective results at the example of Volumetric Depth Images (VDIs), a view-dependent representation for volumetric data. VDIs can quickly and flexibly be generated via a modified volume raycasting procedure that partitions and partially composits samples along view rays. In particular, we study the impact of respective input parameters on this process w.r.t. the involved quality-space trade-off. We quantify rendering quality via image quality metrics and space requirements via the compressed size of the intermediate representation. On this basis, we then automatically determine the parameter settings that yield the best quality under different constraints. We demonstrate the utility of our approach by means of a variety of different data sets, and show that we optimize the achieved results without having to rely on tedious and time-consuming manual tweaking.

Index Terms—In Situ Visualization, Volume Raycasting, Intermediate Representation, Auto-Tuning.

1 Introduction

Storage and network bandwidth constraints increasingly become the limiting factor for overall compute performance. This has significant impact on almost all applications running on larger scale machines, as not only a significant number of operations is processed, also large amounts of data are produced. In situ visualization tackles this issue by processing and thus reducing data directly when it is generated. This allows to create and store visualizations at a much higher resolution than what the simulation node would otherwise be able to store permanently in terms of full raw data (both in terms of spatial and temporal resolution). We concentrate on a popular variant commonly denoted as hybrid in situ visualization in this paper (Fig. 1). Conceptually, it transforms the data locally into an intermediate visualization representation of reduced size, that can then be used to generate renderings a posteriori with some degrees of freedom. This approach can achieve a large reduction of data size with only minor overhead. However, several parameters influence the generation of an intermediate representation (with an impact on image quality and size, among others). Setting these parameters suitably w.r.t. the demands of the application scenario is a hard task that can be tedious to solve manually.

In this work, we propose an approach to auto-tune parameters for the generation of intermediate representations toward different goals. In doing this, we consider the trade-off between (storage) size, generation time, and rendering quality of intermediate representations. In our implementation, we employ Volumetric Depth Images (VDIs) as intermediate representation [1]. They offer an effective view-dependent representation of volumetric data that can be explored interactively using arbitrary camera configurations. Among others, VDIs have been used for the in situ visualization of CFD simulations [2] [3].

In the following, Sec. 2 gives an overview on related work. Sec. 3 reviews the fundamentals of our approach: hybrid in situ visualization and VDIs. On this basis, we then discuss what we consider to be the main contribution of our work:

- we outline our approach for auto-tuning intermediate representations for in situ visualization (Sec. 4),
- we apply the approach to VDIs, formally discussing respective input and output parameters, the involved trade-off, the choice of utility functions to tune for desired characteristics, etc. (Sec. 5)
- we discuss our results by means for different data sets and utility functions in Sec. 6.

We finally conclude our work in Sec. 7.

2 Related Work

In Situ Visualization. In situ visualization basically embeds the entire data analysis and rendering into the data generation process and transfers only partial results to the display node for final compositing [4]. This approach has been shown

• *Steffen Frey and Thomas Ertl are with University of Stuttgart. E-mail: steffen.frey@visus.uni-stuttgart.de*

978-1-4673-9051-4/16/$31.00 ©2016 IEEE

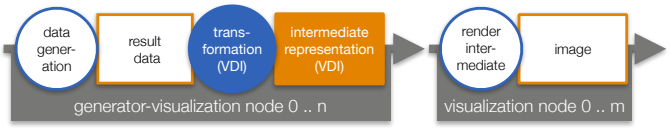
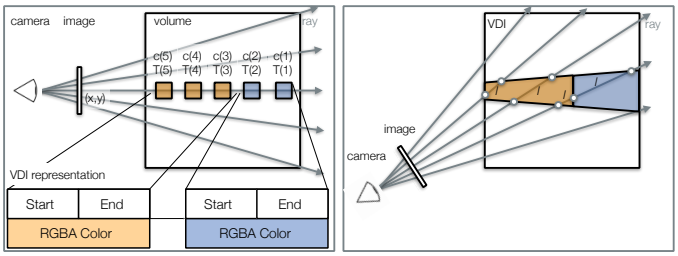


Fig. 1: Hybrid in situ visualization setup: intermediate representations are generated that are later used for rendering.



(a) VDI generation (R). (b) VDI rendering (I).

Fig. 2: VDI generation and data structure. Essentially, samples on are clustered along rays. Only their composited color and depth are stored.

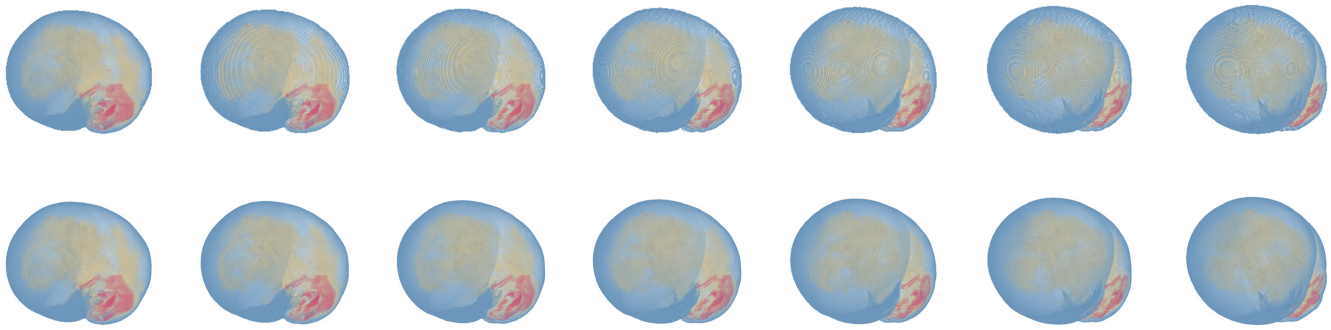
to exhibit good scaling properties across a large number of cores [5]. However, only limited simulation steering and modification of visualization parameters is possible. Bauer et al. [6] present existing methods, infrastructures, and a range of computational science and engineering applications using in situ analysis and visualization. Moreland [7] discusses implementation tradeoffs that result from conflicts with traditional requirements (of domain scientists). Hybrid in situ visualization approaches employ intermediate representations with the goal to significantly reduce the size of the data by transforming it via some visualization steps, yet still preserving flexibility for certain a posteriori adjustments (discussed in more detail in Sec. 3.1). Tikhonova et al. [8] convert a small number of volume renderings to a multi-layered image representation, enabling interactive exploration in transfer function space. In another work, Tikhonova et al. [9] use an intermediate volume data representation based on Ray Attenuation Functions, which encode the distribution of samples along each ray. Vo et al. [10] do an in situ data extraction and perform a more loosely coupled visualization abstraction on a dedicated resource relying on streaming-enabled frameworks. In this work, we employ Volumetric Depth Images (VDIs) as intermediate representation for the efficient representation of volumes [1]. They represent an intermediate abstraction of volume rendering samples, clustered with respect to the compositing along viewing rays. Effectively, VDIs offer an effective representation that can be explored interactively from arbitrary new viewpoints without accessing the original data or simulation. We employ VDIs in this work and discuss it in more detail in Sec. 3.2. The VDI representation has also been extended to Space-Time VDIS (STVDIs) to not only cluster along rays, but also across rays and time steps [2], and has been applied to the visualization of simulations featuring strongly coupled partitioned fluid-structure interaction [3].

Direct Volume Rendering. Direct volume rendering techniques can roughly be classified as image-based, most notably ray-casting, and object-based, such as cell projection, shear-warp, or splatting. Nowadays, GPU-based raycasting [11] is the state-of-the art technique for interactive volume rendering [12]. The generation of VDIs is based on an extended version of a standard GPU raycaster [1]. As an alternative to raycasting, texture-based rendering via 3D textures slices the texture block in back-to-front order with planes oriented parallel to the view plane [13]. In contrast, splatting [14] accumulates data points by projecting flat disc-like kernels for each voxel to the image plane. Many adjustments have been proposed to improve its quality and speed (e.g., [15] [16]). The projected tetrahedra approach renders partially transparent polygons to render volume data [17] [18], based on the projected profile of tetrahedral cells.

Image-based Rendering. Image-based rendering infers new images from existing ones, e.g., with changed lighting or camera configuration [19]. Various techniques have been proposed to construct different representations from multiple views, like view-dependent texture maps [20], warping [21] [22], light fields [23], or Lumigraphs [24]. Meyer et al. [25] employ opacity light fields for image-based volume rendering based on multiple renderings by means of proxy surfaces. Rezk-Salama et al. [26] use depth layers to generate high quality light field representations from volumetric data. Other techniques synthesize new surface-based views from multiple images of volume data [27] [28]. Such techniques allow for adapting color and lighting parameters [29], or transfer functions [30]. Shade et al. [31] proposed LDIs which represent one camera view with multiple pixels along each line of sight. In contrast to VDIs, LDIs and related techniques are targeted toward surfaces (with associated depth values and "vacuum" in between), and not volumes. Multi-layered representations have also been used in commercial rendering software to simulate complex materials like skin on synthetic objects [32]. In volume rendering, layer-based representations allow to defer operations such as lighting and classification [33] [34], or to cache results [35] [36].

Parameter Tuning and Volume Visualization. Automatic parameter tuning evaluates the performance of different settings to find those with the best performance. For volume rendering on the GPU, Bethel and Howison [37] optimize the run time by exploring different variations of raycasting kernels and their execution configuration. Furthermore, a technique has been proposed that flexibly adapts by steering the visualization process in three major degrees of freedom: when to terminate the refinement of a frame in the background and start a new one, when to display a frame currently computed, and how much resources to consume [38]. Another approach has been proposed for directly and dynamically balancing adaptive sampling and compression in remote volume visualization that optimizes the image quality that can be achieved within a prescribed time window [39].

Image Quality Assessment. While the most accurate way to evaluate image quality works via psychophysical experiments with human subjects, these are very expensive, time-consuming and overall infeasible for an automatic tuning process. Software-based quality metrics are employed for a variety of applications, e.g., to monitor video quality, compare the performance of video processing systems and



(a) $0\circ, q = 32.4$ (b) $10\circ, q = 30.8$ (c) $20\circ, q = 30.5$ (d) $30\circ, q = 30.8$ (e) $40\circ, q = 31.1$ (f) $50\circ, q = 31.2$ (g) $60\circ, q = 31.3$

Fig. 3: Supernova (time step 40) rendered via raycasting (bottom) and VDIs (top, $\gamma = 0.013, r = 256$) for different angles.

algorithms, and to optimize the algorithms and parameter settings for a video processing system. In this paper, we use a popular full-reference metric that is commonly used to measure the quality of reconstruction of lossy image and video compression codecs: peak signal-to-noise ratio (PSNR). Full-reference metric means that a (lossy) image is compared against a reference (the "perfect" image). PSNR is a classic and comparably simple metric, that is based on the mean squared error of pixel values. Despite its simplicity, experiments have shown that PSNR works well as long as the content and the codec type are not changed (like in our setup) [40]. Freitas et al. [41] also demonstrated its utility for a wide range of scenarios. Its values are in the range $[0, \infty)$, with 0 standing for maximum difference and ∞ denoting identity. As the reference image is required for each computed frame in our evaluation, we basically run our visualization procedure twice: first using the actual image generated by rendering the intermediate representation, and second, the reference that generates images using the actual data.

For image sequences, a popular full reference metric is MOVIE (MOtion-based Video Integrity Evaluation) by Seshadrinathan and Bovik [42] (it has also been used for auto-tuning volume visualization [38], among others). Other notable metrics include DRIVQM [43], and the software package VQMT, which contains various different metrics (e.g., [44]).

3 Fundamentals

In this section, we review the basics of our approach. First, we look at hybrid in situ visualization (Sec. 3.1), and then we review the fundamentals and properties behind Volumetric Depth Images (VDIs) (Sec. 3.2).

3.1 Hybrid In Situ Visualization

While there are different in situ visualization setups, we concentrate on a popular variant commonly denoted as hybrid in situ visualization in this paper (Fig. 1). Conceptually, we transform the data into an intermediate visualization representation of reduced size on the same node that generated the respective data (*generator-visualization node*). This intermediate representation can then be used for the efficient (and typically interactive) rendering on a separate *visualization node*.

More formally, this means that we have two fundamental functions R and I:

- $R(S_A, P_A, P_M) \rightarrow M$ transforms a scene S_A with a set of (view) parameters P_A into an intermediate representation M. While P_A describes the parameters that define what is seen (like the camera configurations), P_M describes the parameters that specify the generation process of M (like resolution). This is done on *generator-visualization nodes*.
- $I(M, P_I) \rightarrow O$ transforms intermediate representation M with parameters P_I into output image O. This takes place on *visualization nodes*.

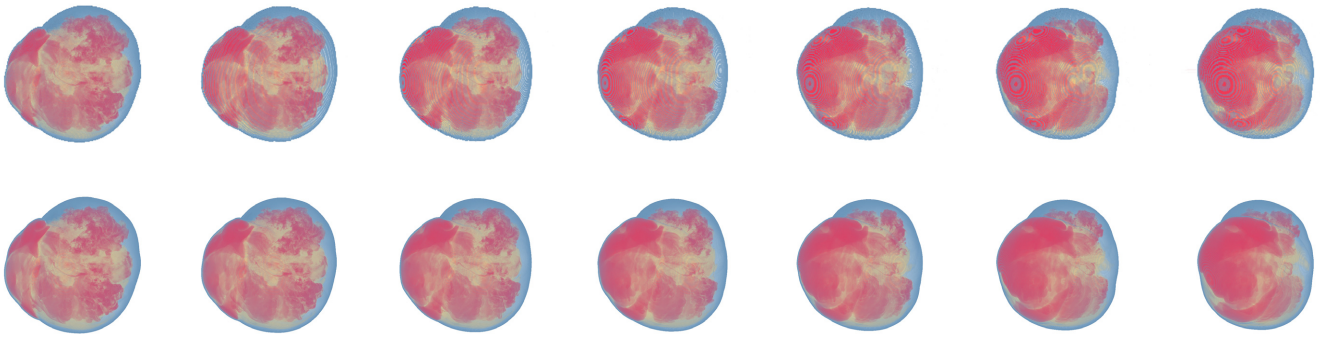
While this approach promises a large reduction of data size with potentially small minor overhead, the appropriate tuning of parameters, particularly P_M, for generating the intermediate representation generation can be tedious to achieve.

3.2 Volumetric Depth Images

Volumetric Depth Images (VDIs) can be seen as a generalization of Layered Depth Images (LDIs) for volume data (cf. Sec. 2). It is a view-dependent representation (M) that allows to capture a volume (S_A) from a certain reference camera configuration (P_A) in a fast and efficient way. This representation can then be rendered afterwards from arbitrary camera configurations (P_I). Fig. 2 gives a schematic overview on the concept behind VDIs (we only give a brief explanation in the following, cf. [1] for details). According with the discussion of intermediate representations for in situ visualization above (Sec. 3.1), the usage of the VDIs is split into two phases:

1) the generation (Fig. 2a) (R),
2) and the rendering (Fig. 2b) (I).

A VDI can easily be generated during volumetric raycasting by partitioning the samples along rays according to their similarity (Fig. 2a). As parameters P_M, we consider the resolution r (depicting a resolution of r^2 in image space, i.e., r^2 rays), and the interrupt parameter γ that basically determines the threshold up until to what color difference



(a) $0\circ, q = 32.8$ (b) $10\circ, q = 31.1$ (c) $20\circ, q = 31.4$ (d) $30\circ, q = 32.5$ (e) $40\circ, q = 32.9$ (f) $50\circ, q = 33.0$ (g) $60\circ, q = 32.8$

Fig. 4: Supernova (time step 20) rendered via raycasting (bottom) and VDIs (top, $\gamma = 0.013, r = 256$) for different angles.

new samples can be merged into an existing cluster along the ray. A lower resolution r naturally leads to lower rendering quality but requires less space, while a lower value for γ leads to a more selective clustering and with this a higher depth resolution (effectively resulting in both increased quality and storage requirements). These parameters γ and r ($\subset P_M$) essentially define the involved trade-off between storage space, generation cost, and rendering quality. The resulting partitions are stored as lists of so-called supersegments containing the bounding depth pair (*Start*, *End*) and partial color accumulation values (*RGBA Color*). Note that empty (i.e., fully transparent) supersegments are not stored explicitly.

For rendering, conceptually, each (quadrilateral) pixel in the image plane with the used camera parameters forms a pyramid. The supersegments can then be rendered as frustums of these pyramids (Fig. 2b). Frustums generated by the same ray are conceptually grouped into a frustum list. Color and opacity are modeled constant within a frustum, and determined during VDI-generation. For rendering, such a frustum can be represented by proxy geometry. During rendering, frustum lists are depth-ordered and composited. For this, the length l that a ray passes through each frustum needs to be considered in order to correctly adjust its opacity contribution.

4 Auto-Tuning Intermediate Representations

The goal of this work is to determine the best parameter settings P_M for generating intermediate representations. For this, we analyze the impact of different parameters on storage size, generation time and the quality of resulting renderings. Our respective approach for auto-tuning is outlined in Alg. 1 and discussed in the following.

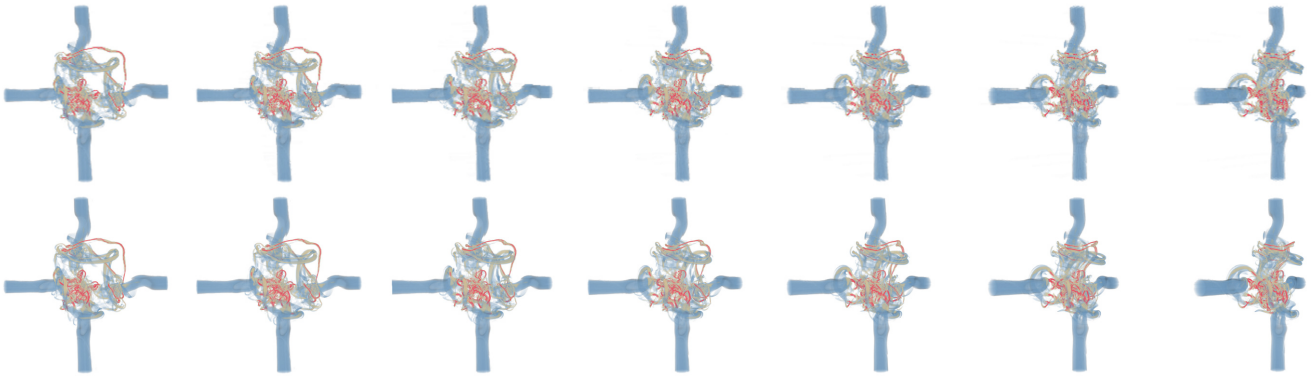
Overall, our approach (i.e., our main function AUTOTUNEREPRESENTATIONS()) basically aims to find the best parameter settings P_M, i.e., the one with the minimal associated utility value $\check{\alpha}$ (Line 25). In the setup phase of our approach, we first set the initial utility value $\check{\alpha}$ to ∞ (Line 2, smaller is better), and choose a scene S_A and an associated parameter set P_R (e.g., camera configuration) that serves as the basis of the measurement series (Line 3). We then also define a series of test parameters $P_I^{[]}$ that are used to evaluate the generated intermediate representations with (Line 4). We then also need to define a utility function

Algorithm 1 Overview on our approach for auto-tuning intermediate representations for in situ visualization.

```
1:  function AUTOTUNEREPRESENTATIONS
2:      α̌ ← ∞
3:      choose scene S_A and parameters P_R
4:      define set of test parameters P_I^[]
5:      define utility function ν
6:      loop
7:          get new parameter set P_M
8:          if P_M = ∅ then
9:              break
10:         end if
11:         τ ← timing[(M ← R(S_R, P_R, P_M))]
12:         σ ← |M|
13:         for all P_I ∈ P_I^[] do
14:             O ← I(M, P_I)
15:             Ǒ ← I(S_A, P_I)
16:             O^[] ← O^[] + (O, Ǒ)
17:         end for
18:         q ← imageQualityMetric(O^[])
19:         α ← ν(σ, q, τ)
20:         if α < α̌ then
21:             α̌ ← α
22:             P̌_M ← P_M
23:         end if
24:     end loop
25:     return P_M, α̌
26: end function
```

$\nu : \mathbb{R}^n \rightarrow \mathbb{R}$ (Line 5). As discussed in more detail below, it provides the basis for optimization as it transforms the various measured performance numbers belonging to an intermediate representation into a single scalar value α that can be used directly for tuning.

On this basis, we then start our main loop for automatic tuning, basically aiming to determine an optimum $\min_{P_M}(\nu(\cdot))$ by choosing parameters P_M accordingly (Lines 6–24). In each iteration, we first obtain a new parameter set P_M that controls the generation of intermediate representations that will be tested in the following (Line 7). If no new parameter can be obtained (Line 8, e.g., when a pre-defined list of parameters has been fully processed),



(a) $0\circ, q = 31.2$ (b) $10\circ, q = 31.0$ (c) $20\circ, q = 31.1$ (d) $30\circ, q = 32.1$ (e) $40\circ, q = 32.3$ (f) $50\circ, q = 32.7$ (g) $60\circ, q = 33.0$

Fig. 5: Renderings of the λ_2 data set via raycasting (bottom) and VDIs (top, $\gamma = 0.001, r = 192$) for different angles.

the optimization loop is exited (Line 9). With P_M, we then generate a new intermediate representation M (Line 11). We also time the generation of M (via timing[·]) to obtain creation cost τ. Additionally, we measure the size of M to assess induced storage and transfer cost (Line 12). We then loop over our set of test parameters $P_I^{[]}$ to generate data for quality assessment (Lines 13–17). In detail, we first generate an output image O with intermediate representation M (Line 14). We then render reference image $\check{O}$ by using the same parameters, but not employing the intermediate representation but the original scene S_A directly (Line 15). These two outputs O and $\check{O}$ are then appended to list $O^{[]}$ (Line 16). This list $O^{[]}$ is then used for assessing quality q with a full-reference quality metric (Line 18).

With this, we then have a tuple of output values (σ, q, τ):

- σ: data size
- q: image quality
- τ: generation time

However, for tuning or optimization, we basically require a single value. We achieve this by applying utility function ν that maps the output tuple to a single (scalar) utility value α (Line 19). If α is smaller than the best value so far $\check{\alpha}$ (Line 20), we update $\check{\alpha}$ (Line 21) as well as the associated parameters $\check{P_M}$ for the generation of the intermediate representation (Line 22).

Finally, the optimization loop exits as soon as no new parameter set is generated (Lines 7–9). Then, the minimal determined utility value $\check{\alpha}$ and the associated parameters $\check{P_M}$ for the generation of intermediate representations are returned from our auto-tuning function (Line 25). In the following, we discuss our implementation of this approach for the optimization of VDIs (Sec. 5).

5 Auto-Tuning VDIs

In this section, we discuss the application of our auto-tuning approach discussed previously in Sec. 4 to VDIs. For a specified camera configuration and transfer function, VDIs represent an intermediate abstraction of color-mapped volume samples. The storage of their respective depth values basically allows for a depth-aware reconstruction of the volume data for rendering (cf. Sec. 3.2). In the following, we discuss the optimization of the performance of VDIs under various constraints. We first look at what an input scenario S looks like in this context (Sec. 5.1), and then discuss the different parameters that may be chosen for generating a VDI (Sec. 5.2). We then discuss how we obtain our tuple of output values (σ, q, τ) (Sec. 5.3), and how we can define different utility functions ν on their basis (Sec. 5.4). This then lays the foundation for our simple auto-tuning approach evaluating different parameter configurations for P_M (Sec. 5.5).

5.1 Input Scenario (S_A and P_A)

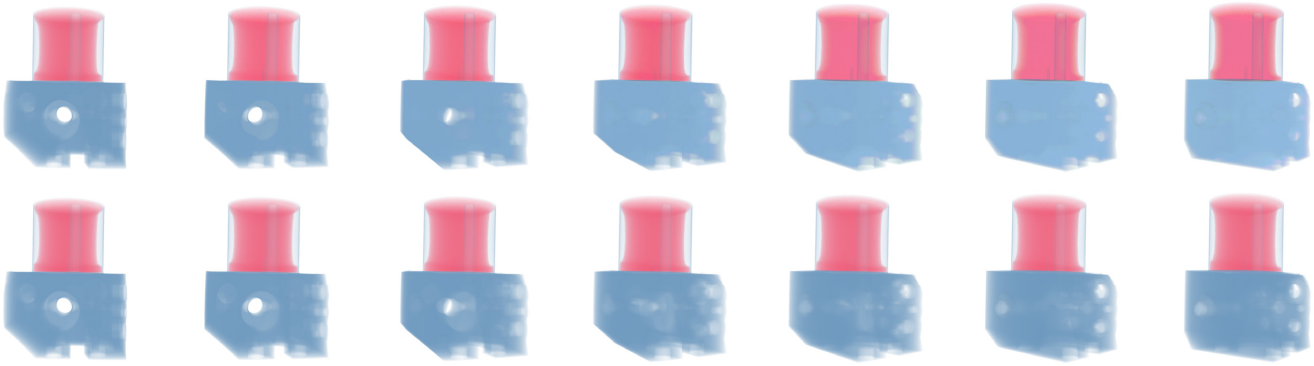
VDIs are a view-dependent representation of volumes. Accordingly, the input scenario basically consists of an ordinary scene and parameter description that could be used as an input to any volume raycaster to generate an image. This can be classified into the following three components:

1) Naturally, the volumetric (scalar) data set needs to be provided. In the following, we primarily focus our discussion on single volumes, but also a (subset of a) whole time series could be considered (cf. discussion in Section6.2) ($\subset S_A$).
2) VDIs collect samples along a ray after the mapping of a transfer function to yield color and opacity. Accordingly, this transfer function also needs to be supplied for VDI generation ($\subset S_A$).
3) VDIs are view-dependent, and therefore generated from a certain camera configuration (cf. Fig. 2a). This basically comprises a camera's position, orientation, and field of view (P_A).

5.2 Input Parameters $P_M \rightarrow (r, \gamma)$

VDIs have two basic parameters, both of which have an impact on the size of resulting representation M, the image quality that can be achieved with it, as well as the generation time:

1) **Resolution r.** The resolution r in image space basically defines the number of rays that are used to generate a VDI.
2) **Interrupt parameter γ.** The interrupt parameter γ in ray space provides the threshold that is used to determine whether a new sample along the ray is a good match for the current supersegment. The



(a) $0\circ, q = \infty$ (b) $10\circ, q = 42.5$ (c) $20\circ, q = 35.8$ (d) $30\circ, q = 32.5$ (e) $40\circ, q = 30.3$ (f) $50\circ, q = 28.7$ (g) $60\circ, q = 27.4$

Fig. 6: Renderings of the Zeiss data set via raycasting (bottom) and VDIs (top, $\gamma = 0.001, r = 768$) for different angles.

employed criterion is based on premultiplied color values and the correction of opacity with respect to integration lengths along a ray. A sample is then merged into the supersegment if the adjusted color and opacity difference is below γ. Otherwise, a new supersegment is started (cf. Fig. 2a, cf. [1] for details).

These parameters basically define the quality-cost trade-off of a VDI, and are therefore the subject of tuning in the following

5.3 Output Performance Indicators (σ, q, τ)

As described in Sec. 4, we consider three different types of performance indicators in this paper. In the following, we discuss how we actually evaluate them in our implementation.

- **Storage Cost σ.** For quantifying the storage cost, we use the compressed size of the VDI representation. For this, we apply bzip2 to the binary storage format of a VDI (Alg. 1, Line 12).
- **Rendering Quality q.** We further evaluate the quality achieved when rendering with different camera configurations P_A. For this, we use the popular peak signal-to-noise ratio (PSNR). PSNR is a full-reference metric that is computed on the basis of the mean squared error (MSE). Accordingly, it determines the quality by comparing a candidate against the reference. In our implementation, for each image O generated with the VDI representation M, we generate a respective reference image $\check{O}$ using direct volumetric raycasting with the original data (Alg. 1, Lines 13–18).
- **Generation Time τ.** Here, we simply measure the time the VDI generation process takes. For this, we employ kernel timers to obtain the execution time of our CUDA-based raycaster on the GPU (Alg. 1, Line 11).

5.4 Utility Evaluation ν

Theoretically arbitrary utility functions ν can be specified to transform our value tuple of performance indicators (σ, q, τ) into a single scalar for minimization (i.e., smaller values are better). In general, their definition depends on the application use case and certain constraints induced by the overall setup. For our evaluation in this paper, we consider two types of such constraints, as discussed below.

First, we tune with respect to different size limitations σ_{target} regarding M, while optimizing the achieved quality. This is classic for many in situ scenarios, in which limiting the data size for storage and transfer is a major concern. For this, our utility function ν_{size} is defined as follows:

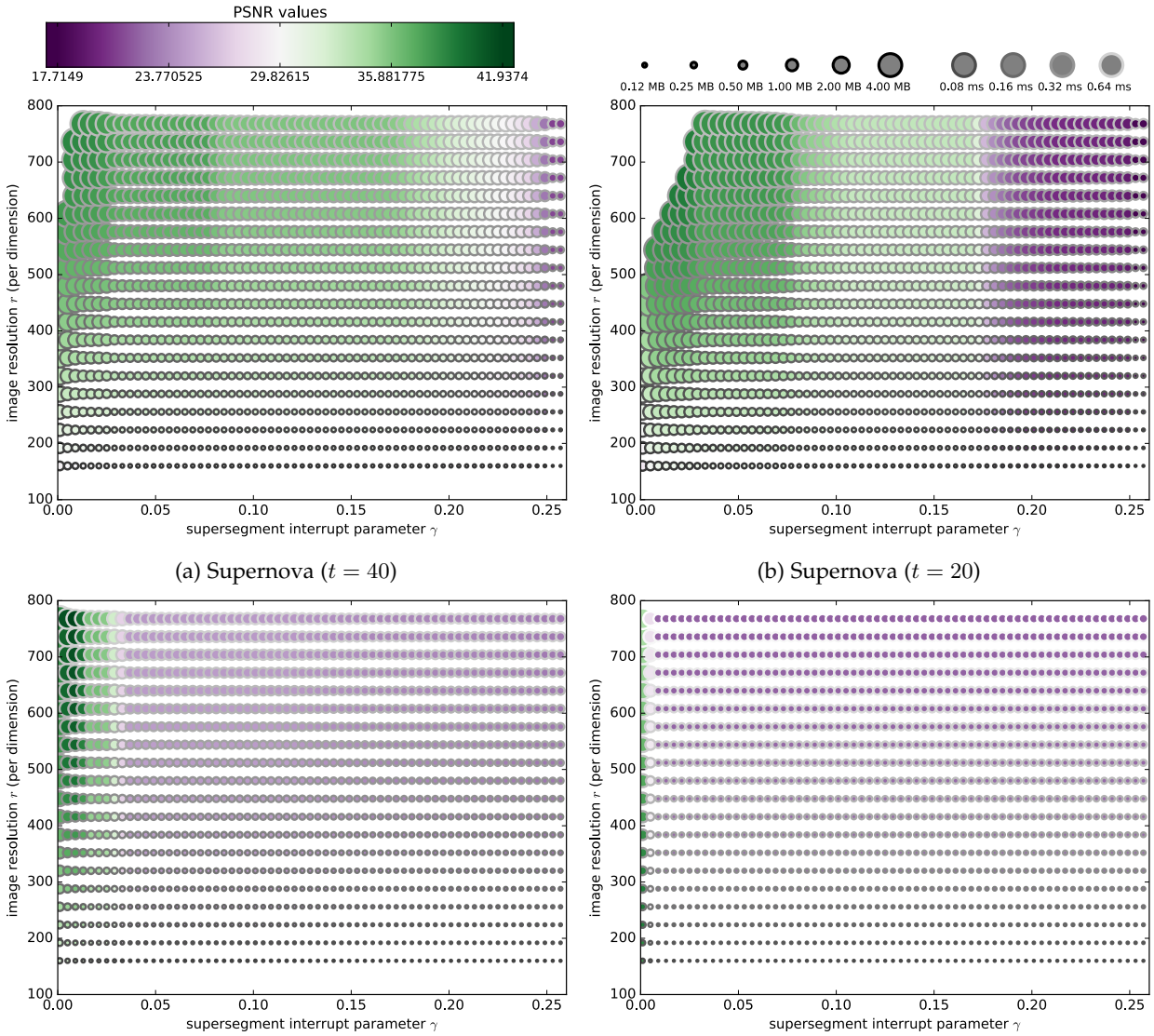
$$\nu_{\text{size}}(\sigma, q, \tau) = \begin{cases} -q & \text{if } \sigma < \sigma_{\text{target}} \\ \infty & \text{else} \end{cases}. \tag{1}$$

Second, we consider the generation time (τ_{target}) as the main quantity of interest. This is important for in situ scenarios in which simulation or measurement output data is produced rapidly, and the visualization needs to be able to keep up. Another application case here would be scheduling in the case of shared compute resources. Here, we use the following utility function ν_{time} to account for this:

$$\nu_{\text{time}}(\sigma, q, \tau) = \begin{cases} -q & \text{if } \tau < \sigma_{\text{time}} \\ \infty & \text{else} \end{cases}. \tag{2}$$

5.5 Tuning

It can be expected that our input parameters γ and r yield higher quality when they are decreased and increased, respectively. However, the rate at which they do essentially depends on the structure and characteristics of the underlying volume data (Alg. 1, Line 7). Therefore this behavior is hard to predict a priori, and we employ a parameter range for both interrupt parameter γ (0.001-0.26, step size 0.004) and resolution r (160-768, step size 32, i.e., an image-space resolution of r^2) to assess it. The respective ranges were determined in a priori experiments. For each obtained result, we compute our performance indicators, evaluate the respective utility function ν to yield α, and compare it to our currently best solution $(\check{\alpha}, \check{P_M})$. If our new parameters lead to an improvement (i.e., $\alpha < \check{\alpha}$), we update our parameter set $\check{P_m}$ (Line 20–22). Note that in our implementation, we first collect and cache all result data, and then do the tuning afterwards. An integrated scheme could also involve adaptive refinement to the respective goal, which has both



(a) Supernova ($t = 40$)

(b) Supernova ($t = 20$)

(c) λ_2

(d) Zeiss

Fig. 7: Measurement series for different data sets with a range of values for r and γ. The charts depict resulting generation time τ (edge brightness of circle), storage space σ (circle area) and quality q (fill color of circle). These values are later used for tuning (missing values caused by VDI proxy geometry exceeding GPU memory for rendering).

the potential to improve accuracy as well as to decrease the overall computation time of the tuning, at least when tuning with one specific utility function ν that is known and fixed beforehand.

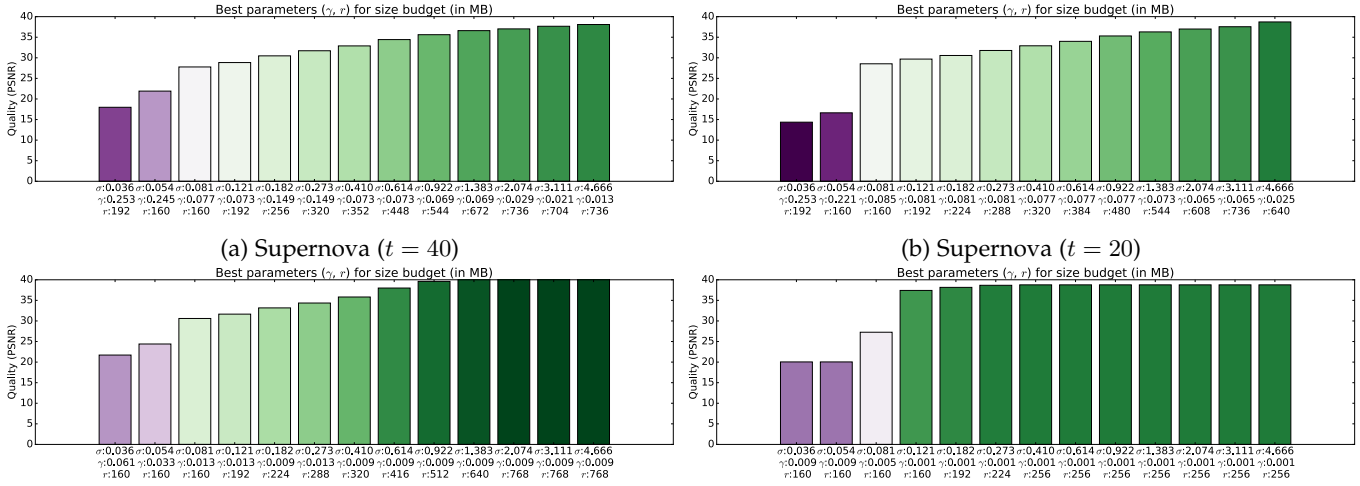
6 Results

In this section, we discuss the results that we achieved by running the approach introduced in this paper. We run our implementation on a machine with an NVIDIA GTX980, an Intel Core i7-3820, and 16GB of memory. VDIs are both generated and rendered using the GPU. We use a CUDA-based front-to-back volume raycaster for VDI generation employing one GPU thread per ray. VDIs are rendered using rasterization as directly supported by commodity graphics hardware using OpenGL and GLSL [1]. The image resolution of all renderings throughout this paper and in the evaluation is 768^2.

For our evaluation, we use the following data sets from different sources:

- **Supernova**. Simulation of a supernova (resolution: 432^3). We consider two time steps of the simulation in the following (Figs. 3 (time step 40) and 4 (time step 20), obtained from vidi.cs.ucdavis.edu). The same transfer function is used for both.
- $\boldsymbol{\lambda_2}$. λ_2 vortex extraction criterion applied to the result of a CFD simulation (Fig. 5, resolution: 529^3).
- **Zeiss.** CT scan of a mechanical component (Fig. 6, resolution: 680^3, courtesy of Daimler AG).

In the following, we first look at the impact of our parameters $P_M = (\gamma, r)$ on the performance indicators (σ, q, τ) (Sec. 6.1). On this basis, we then evaluate the results for tuning with different utility functions ν (Sec. 6.2).



(a) Supernova ($t = 40$)

(b) Supernova ($t = 20$)

(c) λ_2

(d) Zeiss

Fig. 8: Best parameter setting pair (γ, r) with which the specified size limitation σ_{target} can be undercut with the best possible quality (both bar height and colors depict PSNR quality).

source	γ	r	σ	q	τ
$\sigma_{\text{target}} = 0.05$ MB					
40 (from 40)	0.245	160.0	0.053	21.911	0.077
20 (from 40)	0.245	160.0	0.044	16.316	0.075
40 (from 20)	0.221	160.0	0.064	25.695	0.077
20 (from 20)	0.221	160.0	0.052	16.633	0.078
$\sigma_{\text{target}} = 0.08$ MB					
40 (from 40)	0.077	160.0	0.075	27.777	0.078
20 (from 40)	0.077	160.0	0.100	28.915	0.077
40 (from 20)	0.085	160.0	0.072	27.617	0.078
20 (from 20)	0.085	160.0	0.077	28.546	0.076
$\sigma_{\text{target}} = 0.12$ MB					
40 (from 40)	0.073	192.0	0.109	28.845	0.098
20 (from 40)	0.073	192.0	0.169	30.026	0.099
40 (from 20)	0.081	192.0	0.106	28.731	0.098
20 (from 20)	0.081	192.0	0.117	29.691	0.099
$\sigma_{\text{target}} = 0.18$ MB					
40 (from 40)	0.149	256.0	0.165	30.474	0.154
20 (from 40)	0.149	256.0	0.159	29.656	0.155
40 (from 20)	0.081	224.0	0.144	29.724	0.124
20 (from 20)	0.081	224.0	0.159	30.565	0.125
$\sigma_{\text{target}} = 0.27$ MB					
40 (from 40)	0.149	320.0	0.257	31.714	0.216
20 (from 40)	0.149	320.0	0.249	30.381	0.216
40 (from 20)	0.081	288.0	0.237	31.393	0.184
20 (from 20)	0.081	288.0	0.262	31.782	0.183
$\sigma_{\text{target}} = 0.41$ MB					
40 (from 40)	0.073	352.0	0.367	32.892	0.250
20 (from 40)	0.073	352.0	0.570	33.754	0.249
40 (from 20)	0.077	320.0	0.299	32.276	0.211
20 (from 20)	0.077	320.0	0.400	32.918	0.214
$\sigma_{\text{target}} = 0.61$ MB					
40 (from 40)	0.073	448.0	0.593	34.429	0.371
20 (from 40)	0.073	448.0	0.922	35.208	0.371
40 (from 20)	0.077	384.0	0.431	33.341	0.286
20 (from 20)	0.077	384.0	0.576	34.001	0.286
$\sigma_{\text{target}} = 0.92$ MB					
40 (from 40)	0.069	544.0	0.885	35.617	0.517
20 (from 40)	0.069	544.0	1.520	36.464	0.516
40 (from 20)	0.077	480.0	0.674	34.783	0.417
20 (from 20)	0.077	480.0	0.901	35.309	0.415

TABLE 1: Cross-evaluation of results for the different parameter sets obtained from tuning for time steps 20 and 40 of the Supernova (for σ_{target}, respective tuning results are listed in Fig. 8a and b).

6.1 Parameter Study

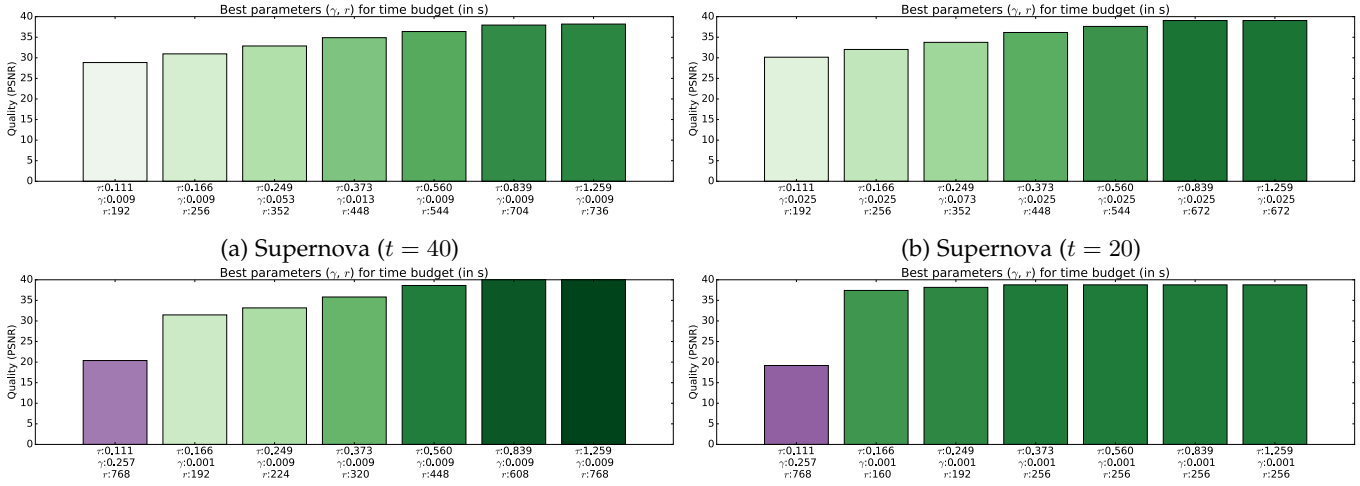
First, we consider camera view angles from $10°$ to $60°$ (in steps of $10°$) around the y-axis to define characteristic extents of deviation for visual exploration w.r.t. a given task. For the different data sets, the respective results are shown in Fig. 7. Generally, as can be expected, smaller r and larger γ lead to worse image quality, however they are also able to deliver smaller data sizes. It can also be seen that the generation time primarily depends on the resolution r, naturally increasing with higher resolution as more rays are sent for VDI generation.

As VDIs are inherently a data-dependent representation, its resulting characteristic values can differ significantly across different data sets. While the two time steps from the Supernova simulation (a and b) are fairly similar overall (although a higher visual complexity of time step 40 w.r.t. time step 20 can be seen already), there are significant differences to λ_2 (c) and Zeiss (d). In general, the resolution r has a comparably similar impact on quality, size, and generation time, yet the impact of interrupt parameter γ varies hugely. The reason for that is that particularly for data sets with clear boundaries that are similar across the whole data set (like the Zeiss), there is one crucial step in the parameter range that defines how these boundaries are handled (d, in the range of $\gamma = 0.1$–$\gamma = 0.5$). In contrast, the transition is relatively smooth for the Supernova data set (a and b).

It can also been from the plot already that some parameter configurations seem to be better-suited than others. For instance, for the Supernova (a), (r=768, γ=0.25) yields approximately the same data size yet much worse quality than (r=608, γ=0.11). These suitability for different parameter configurations will be considered in the following in the context of tuning in Sec. 6.2.

6.2 Tuning Different Data Sets

We now do auto-tuning based on the the results of our parameter study discussed earlier in Sec. 6.1. First, we tune



(a) Supernova ($t = 40$)

(b) Supernova ($t = 20$)

(c) λ_2

(d) Zeiss

Fig. 9: Best parameter setting pair (γ, r) that is able to undercut the specified time limitation τ_{target} yielding the best possible quality (both bar height and colors depict PSNR quality).

for hitting different size limitations σ_{target}, while optimizing the achieved quality. This means that we use the utility function ν_{size} defined in Eq. 1. The respective results for different data sets are shown in Fig. 8. It shows the respective results for a range of different storage constraints (each limit is 1.5 times larger than the one of its predecessor). Overall, it can be seen, that the more storage space is available, the higher is the rendering quality that can be achieved. In more detail, regarding the determined best solutions from a low to a high storage budget, Fig. 8 shows that essentially both γ and r develop towards higher quality, but the rate at which they do essentially depends on the structure and characteristics of the data, and is therefore hard to predict in general and would be tedious to manually tune.

Next, we also look at results that can be achieved when generation time (τ_{target}) is of importance with utility function ν_{time} (Eq. 2). The respective results are shown in Fig. 9. Like for data size restrictions, it can be seen that the more time is available for VDI generation, the better the achieved quality can get across all data sets.

Finally, we aim to look a the tuning results w.r.t. multiple steps of a time series at the example of the two Supernova time steps. For this, we look at the results in Table 1. It can be seen that parameter settings tuned with respect to the one time step also translate to the other time step to a certain extent, yet obviously no guarantees can be given without explicit consideration, and the how well it works basically depends on their similarity. Therefore, when optimizing for a time series, not just one time step should be used, but a couple of characteristic time steps should be picked if staying within certain bounds is of high importance. In the simplest approach, when looking at the list of scenes, one could just use the most conservative determined parameter settings that stays within the bounds for each of them. However, the detailed consideration and discussion of time series is beyond the scope of this paper and remains for future work.

7 Conclusion

We presented our approach to analyze the impact of input parameters on intermediate representations for in situ visualization, and to auto-tune them regarding different constraints. We demonstrated its application and results by employing VDIs, a view-dependent representation for volumetric data. We studied the impact of parameters on the quality-space trade-off of different data sets, and on this basis automatically determined the settings that yield the best quality results for different storage and timing constraints. We could show that this allows to optimize the results that can be achieved, without requiring manual tweaking.

There are several possible directions for future work. While storage space and computation time on simulation nodes are arguably the most common restriction in practice, there are also numerous other factors of interest, like energy consumption. We also only considered one intermediate representation in the discussion above (VDIs), yet auto-tuning might also be applied to choose between different representations taking into account the required degrees of freedom for a posteriori analysis, as well as reduction in storage and computational cost. While in our implementation, we first collect and cache all result data, and then do the tuning afterwards, an integrated, adaptive scheme could improve accuracy and decrease total computation time, at least when tuning with a pre-defined utility function ν. Finally, when optimizing for a time series, not just one but a collection of characteristic time step should be used for tuning.

Acknowledgments

The authors would like to thank the German Research Foundation (DFG) for supporting the project within project A02 of SFB/Transregio 161 and the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

REFERENCES

[1] S. Frey, F. Sadlo, and T. Ertl, "Explorable volumetric depth images from raycasting," in *26th Conference on Graphics, Patterns and Images*, 2013, pp. 123–130.

[2] O. Fernandes, S. Frey, F. Sadlo, and T. Ertl, "Space-time volumetric depth images for in-situ visualization," in *IEEE Large Data and Visualization 2014 (LDAV14)*, 2014.

[3] O. Fernandes, D. S. Blom, S. Frey, A. H. Van Zuijlen, H. Bijl, and T. Ertl, "On in-situ visualization for strongly coupled partitioned fluid-structure interaction," in *Coupled Problems 2015: Proceedings of the 6th International Conference on Computational Methods for Coupled Problems in Science and Engineering, Venice, Italy, 18-20 May 2015*. CIMNE, 2015.

[4] K.-L. Ma, "In situ visualization at extreme scale: Challenges and opportunities," *IEEE Computer Graphics and Applications*, vol. 29, no. 6, pp. 14–19, 2009.

[5] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. Petersen, "An image-based approach to extreme scale in situ visualization and analysis," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 424–434. [Online]. Available: http://dx.doi.org/10.1109/SC.2014.40

[6] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, and E. W. Bethel, "In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms," *Computer Graphics Forum*, 2016.

[7] K. Moreland, "The tensions of in situ visualization," *IEEE Comput. Graph. Appl.*, vol. 36, no. 2, pp. 5–9, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1109/MCG.2016.35

[8] A. Tikhonova, C. Correa, and K.-L. Ma, "Explorable images for visualizing volume data," in *IEEE Pacific Visualization Symposium*, 2010, pp. 177–184.

[9] A. Tikhonova, C. D. Correa, and K.-L. Ma, "An exploratory technique for coherent visualization of time-varying volume data," in *Eurographics conference on Visualization*. Eurographics Association, 2010, pp. 783–792.

[10] H. Vo, J. Comba, B. Geveci, and C. Silva, "Streaming-enabled parallel data flow framework in the visualization toolkit," *IEEE Computing in Science & Engineering*, vol. 13, no. 3, pp. 14–19, 2011.

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

[12] C. Rezk-Salama, M. Hadwiger, T. Ropinski, and P. Ljung, "Advanced illumination techniques for gpu volume raycasting," in *ACM SIGGRAPH Courses Program*, 2009.

[13] J. Krüger and R. Westermann, "Acceleration techniques for gpu-based volume rendering," in *IEEE Visualization*, 2003, pp. 287–292.

[14] L. A. Westover, "Splatting: a parallel, feed-forward volume rendering algorithm," Ph.D. dissertation, 1991, uMI Order No. GAX92-08005.

[15] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, "High-quality splatting on rectilinear grids with efficient culling of occluded voxels," *IEEE Trans. on Visualization and Computer Graphics*, vol. 5, no. 2, pp. 116–134, 1999.

[16] F. Vega-Higuera, P. Hastreiter, R. Fahlbusch, and G. Greiner, "High performance volume splatting for visualization of neurovascular data," in *IEEE Visualization*, 2005, pp. 271–278.

[17] P. Shirley and A. Tuchman, "A polygonal approximation to direct scalar volume rendering," in *Computer Graphics*, 1990, pp. 63–70.

[18] S. Röttger, M. Kraus, and T. Ertl, "Hardware-accelerated volume and isosurface rendering based on cell-projection," in *IEEE Visualization*, 2000, pp. 109–116.

[19] H.-Y. Shum and S. B. Kang, "A survey of image-based rendering techniques," in *Videometrics, SPIE*, 1999, pp. 2–16.

[20] P. E. Debevec, C. J. Taylor, and J. Malik, "Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach," in *23rd annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '96, 1996, pp. 11–20.

[21] W. R. Mark, L. McMillan, and G. Bishop, "Post-rendering 3d warping," in *Symposium on Interactive 3D graphics*, 1997, pp. 7–16.

[22] L. McMillan and G. Bishop, "Plenoptic modeling: an image-based rendering system," in *22nd annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '95, 1995, pp. 39–46.

[23] M. Levoy and P. Hanrahan, "Light field rendering," in *23rd annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '96, 1996, pp. 31–42.

[24] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," in *23rd annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '96, 1996, pp. 43–54.

[25] M. Meyer, H. Pfister, C. Hansen, C. Johnson, M. Meyer, H. Pfister, C. Hansen, and C. Johnson, "Image-based volume rendering with opacity light fields," no. UUSCI-2005-002. Tech Report, 2005.

[26] C. Rezk-Salama, S. Todt, and A. Kolb, "Raycasting of light field galleries from volumetric data," in *Eurographics conference on Visualization*, ser. EuroVis'08. Eurographics Association, 2008, pp. 839–846.

[27] J.-J. Choi and Y.-G. Shin, "Efficient image-based rendering of volume data," in *Computer Graphics and Applications, 1998. Pacific Graphics '98. Sixth Pacific Conference on*, 1998, pp. 70–78, 226.

[28] B. Chen, A. Kaufman, and Q. Tang, "Image-based rendering of surfaces from volume data," in *Eurographics conference on Volume Graphics*, ser. VG'01. Eurographics Association, 2001, pp. 281–300.

[29] T. He, L. Hong, A. Kaufman, and H. Pfister, "Generation of transfer functions with stochastic search techniques," in *IEEE Visualization*, 1996, pp. 227–234.

[30] Y. Wu and H. Qu, "Interactive transfer function design based on editing direct volume rendered images," *IEEE Trans. on Visualization and Computer Graphics*, vol. 13, no. 5, pp. 1027–1040, 2007.

[31] J. Shade, S. Gortler, L.-w. He, and R. Szeliski, "Layered depth images," in *25th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '98, 1998, pp. 231–242.

[32] C. Donner and H. W. Jensen, "Light diffusion in multi-layered translucent materials," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1032–1039, 2005.

[33] T. Ropinski, J. Prassni, F. Steinicke, and K. Hinrichs, "Stroke-based transfer function design," in *Fifth Eurographic, IEEE VGTC conference on Point-Based Graphics*, ser. SPBG'08. Eurographics Association, 2008, pp. 41–48.

[34] P. Rautek, S. Bruckner, and E. Gröller, "Semantic layers for illustrative volume rendering," *IEEE Trans. on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1336–1343, 2007.

[35] E. J. Luke and C. D. Hansen, "Semotus visum: a flexible remote visualization framework," in *IEEE Visualization*, ser. VIS '02, 2002, pp. 61–68.

[36] E. LaMar and V. Pascucci, "A multi-layered image cache for scientific visualization," in *2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, ser. PVG '03, 2003, pp. 61–68.

[37] E. W. Bethel and M. Howison, "Multi-core and many-core shared-memory parallel raycasting volume rendering optimization and tuning," *International Journal on High Performance Computing Appllications*, vol. 26, no. 4, pp. 399–412, Nov. 2012. [Online]. Available: http://dx.doi.org/10.1177/1094342012440466

[38] S. Frey, F. Sadlo, K.-L. Ma, and T. Ertl, "Interactive progressive visualization with space-time error control," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2397–2406, 2014.

[39] S. Frey, F. Sadlo, and T. Ertl, "Balanced sampling and compression for remote visualization," in *SIGGRAPH Asia 2015 Visualization in High Performance Computing*, ser. SA '15. New York, NY, USA: ACM, 2015, pp. 1:1–1:4. [Online]. Available: http://doi.acm.org/10.1145/2818517.2818529

[40] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of PSNR in image/video quality assessment," *Electronics Letters*, vol. 44, no. 13, pp. 800–801, 2008.

[41] R. de Freitas Zampolo and R. Seara, "A comparison of image quality metric performances under practical conditions," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 3, 2005, pp. III–1192–5.

[42] K. Seshadrinathan and A. C. Bovik, "Motion tuned spatio-temporal quality assessment of natural videos," *IEEE Transactions on Image Processing*, vol. 19, no. 2, pp. 335–350, 2010. [Online]. Available: http://dx.doi.org/10.1109/TIP.2009.2034992

[43] T. O. Aydin, M. Čadík, K. Myszkowski, and H.-P. Seidel, "Video quality assessment for computer graphics applications," in *Proceedings of ACM SIGGRAPH Asia 2010*, ser. SIGGRAPH ASIA '10, 2010, pp. 161:1–161:12. [Online]. Available: http://doi.acm.org/10.1145/1866158.1866187

[44] H. Sheikh and A. Bovik, "Image information and visual quality," *IEEE Transactions on Image Processing*, vol. 15, no. 2, pp. 430–444, 2006.