Interactive Progressive Visualization with Space-Time Error Control

Steffen Frey, Filip Sadlo, *Member, IEEE*, Kwan-Liu Ma, *Fellow, IEEE*, and Thomas Ertl, *Member, IEEE*

Abstract—We present a novel scheme for progressive rendering in interactive visualization. Static settings with respect to a certain image quality or frame rate are inherently incapable of delivering both high frame rate for rapid changes and high image quality for detailed investigation. Our technique 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. We base these decisions on the correlation of the errors due to insufficient sampling and response delay, which we estimate separately using fast yet expressive heuristics. To automate the configuration of the steering behavior, we employ offline video quality analysis. We provide an efficient implementation of our scheme for the application of volume raycasting, featuring integrated GPU-accelerated image reconstruction and error estimation. Our implementation performs an integral handling of the changes due to camera transforms, transfer function adaptations, as well as the progression of the data to in time. Finally, the overall technique is evaluated with an expert study.

Index Terms—Progressive visualization, error-based frame control, interactive volume raycasting.

1 Introduction

The complexity of phenomena that need to be analyzed using visualization techniques increases at a rapid pace. Impressive advances have been made in the last decades to deal with this by means of new and improved visual representations as well as performance optimization. However, in many cases, the demands for both responsiveness and high quality rendering in interactive visualization can still not be satisfied with the available compute resources. Since science and engineering typically involve applications that are at the limit or even beyond the capabilities of available hardware, considerable effort is put into the development of strategies that shift this limit with respect to certain needs. One widely pursued approach is optimization "from the problem side", e.g., by making use of techniques that are able to efficiently model desired aspects of the investigated phenomena. Despite their wide success in graphics and visualization, these techniques have the drawback that they are specific to the investigated problem, reducing versatility, and potentially introducing significant preprocessing overhead. Another approach is "from the observer side", e.g., by quantization and discretization with respect to image-space sampling. As an example thereof, progressive rendering continuously enhances a view when the exact rendering would take too long to compute.

Despite these efforts, achieving a fluent, interactive user experience during exploration is challenging, as it needs to fulfill several requirements, like fast response to user input, and adequate rendering quality. It should further avoid repeated manual parameter tuning, yet still provide degrees of freedom to accommodate user preferences. The traditional approach to handle the inherent trade-off by choosing a fixed setting with respect to frame rate or quality has significant shortcomings in many situations (Fig. 1). Choosing a good setting and maintaining it during a visualization session can impede proper analysis. This difficulty is not limited to image quality and frame rate—it also applies to compute resource utilization, which is a major factor in a variety of scenarios, e.g., due to its impact on power consumption in mobile or supercomputing environments.

In this work, after reviewing related work (Sec. 2) and formulating a generic model of progressive visualization (Sec. 3), we propose and evaluate a dynamic model to optimize the efficiency of flexible progressive rendering in visualization. Our contributions include:

- We introduce a model for dynamic minimization of spatio-temporal error and resource utilization in progressive visualization (Sec. 4), which can easily be integrated with existing systems.
- For this model, we present an approach for automatic parameter optimization using video quality metrics (Sec. 4.4).
- We provide an efficient implementation of the model for progressive volume raycasting that can handle different types of interactive changes in an integral and expressive manner (Sec. 5).
- We systematically demonstrate the utility of dynamically adjusting the rendering cost for good spatio-temporal behavior. The evaluation employs a video metric and an expert study (Sec. 6).

Finally, Sec. 7 provides a conclusion and an outlook to future work.

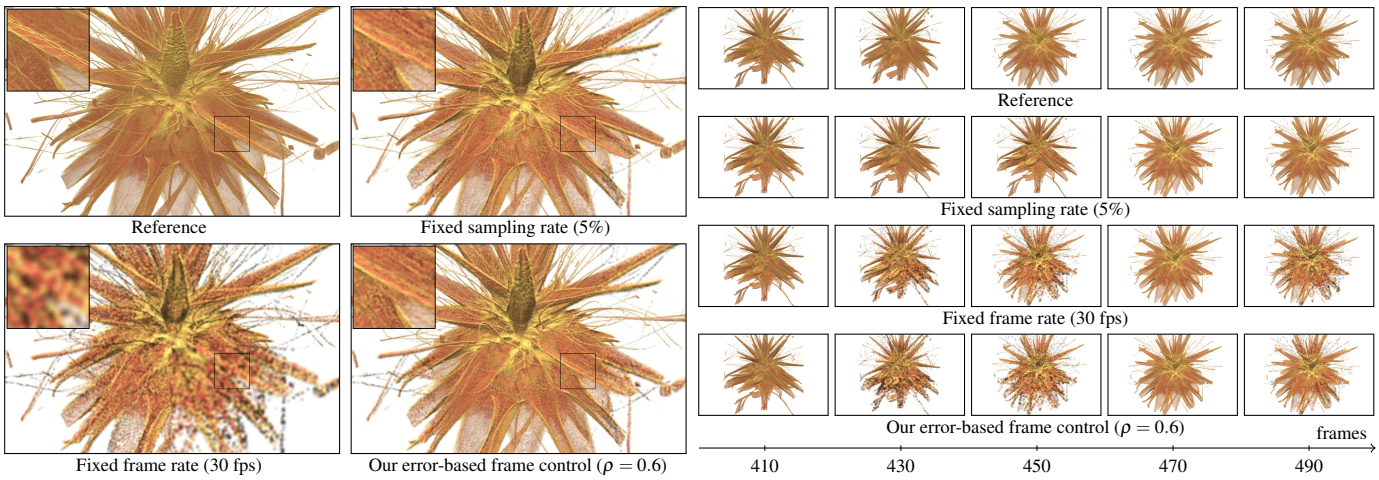
- *Steffen Frey, Filip Sadlo, and Thomas Ertl are with University of Stuttgart. E-mail: {steffen.frey, filip.sadlo, thomas.ertl}@visus.uni-stuttgart.de.*
- *Kwan-Liu Ma is with UC Davis. E-mail: klma@ucdavis.edu.*

Manuscript received 31 Mar. 2014; accepted 1 Aug. 2014; date of publication xx xxx 2014; date of current version xx xxx 2014.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

2 Related Work

The importance of frame rate and quality has been shown in several studies for both video and interactive applications [9, 15]. In general, both factors are equally significant for user experience and performance, as concluded, for instance, by Claypool and Claypool [10] from a user study featuring over 25 participants that evaluated the impact of frame rate and resolution in the context of video games. In this paper, we conduct a user study with visualization researchers to compare the suitability of different methods and determine good parameter settings for our dynamic steering. User studies in visualization have recently been reviewed by Isenberg et al. [16]. Tory and Möller [33] discuss human factors in user studies and visualization design. Anderson et al. [1] explore brain activity recorded using electroencephalography (EEG) to compare different visualization techniques by means of the impact on a viewer's cognitive resources. Notable uses of user studies in scientific visualization include the evaluation of perception for common techniques in uncertainty visualization [29]. Laidlaw et al. [20] compare visualization methods for two-dimensional vector field data by means of simple yet representative tasks.

To achieve good results with our volume raycaster even with only few samples, we employ an image-space sampling scheme based on the capacity-constrained point distributions by Balzer et al. [4], and Gaussian filters for reconstruction. Early image-space acceleration techniques for volume raycasting were presented by Levoy (e.g., casting one ray for multiple pixels [21]). Kratz et al. [19] use an error estimator from the field of finite element methods for adaptive screen-space sampling. For unstructured volumes, Callahan and Silva [8] propose to employ the combination of a low-resolution image of the whole data set and a high-resolution image of the boundary geometry. In contrast to these techniques, which consider the current frame



(a) Slow movement: strong loss of detail for fixed frame rate (frame 1280 shown).

(b) Rapid movement: severe lag for fixed sampling rate (e.g., at frame 450).

Fig. 1: In interactive visualization, different sampling rate or frame settings are desirable for different user actions (see Sec. 5.1 for our definition of sampling rate). In contrast to static frame rate or static sampling rate settings, our error-based approach adjusts dynamically to optimize the utility for a user. The frame numbers are given with respect to the video used for automatic evaluation in Sec. 6.2 (30 frames per second).

only, Qu et al. [25] and Shen and Johnson [32], among others, exploit frame coherence by reutilizing pixel values from the previous frame by warping positions to the current frame. Frey et al. [13] propose volumetric depth images, a view-dependent representation that can be generated quickly and allows the rendering of frames from arbitrary camera configurations at low cost for computation and storage.

In contrast to the image-space acceleration techniques for raycasting that are geared toward good quality for undersampling, Monte Carlo techniques for photorealistic rendering mostly employ oversampling that can also be steered adaptively. Overbeck et al. [22] dynamically adjust the distribution of samples to reduce the variance in wavelet space. However, several key points of their algorithm require separate tuning of quality and speed parameters. Also for global illumination, Farrugia and Péroche [12] present a perceptually-based approach for progressive rendering. Bolin and Meyer [7] employ perceptually-based adaptive sampling based on an extended image processing vision model. Rousselle et al. [28] present a greedy sampling strategy for Monte Carlo rendering. Ramasubramanian et al. [26] utilize a physical error metric in global illumination algorithms.

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 [5] optimize the run time by exploring different variations of raycasting kernels and their execution configuration. In this work, we use a video quality metric to assess performance, both for the purpose of evaluation and automatic parameter tuning. Video quality metrics are commonly employed to monitor video quality, compare the performance of video processing systems and algorithms, or optimize algorithms and parameter settings for video processing systems. We use MOVIE (MOtion-based Video Integrity Evaluation) by Seshadrinathan and Bovik [30], whose source code is publicly available for research purposes. It is classified as a full reference metric, i.e., it compares a video to its respective reference. Other notable metrics include DRIVQM [3], and the software package VQMT, which contains a variety of different metrics (e.g., [31]).

In our model, one of the considered factors is resource utilization of the computation device (GPUs in our case). This aspect has been of significantly rising interest in recent years not only for mobile devices but also for supercomputers. Johnsson et al. [18] thoroughly evaluate the relation between rendering algorithms and power consumption. Pool et al. [24] influence the power consumption by adjusting the precision of computations in pixel shader cores. Ribble [27] suggests to limit the frame rate to a minimum and emphasizes the importance of continuing to optimize the code of an application even if the frame rate goal has been reached. For scientific computing applications, Huang et al. [14] evaluate the energy efficiency of GPUs against CPUs.

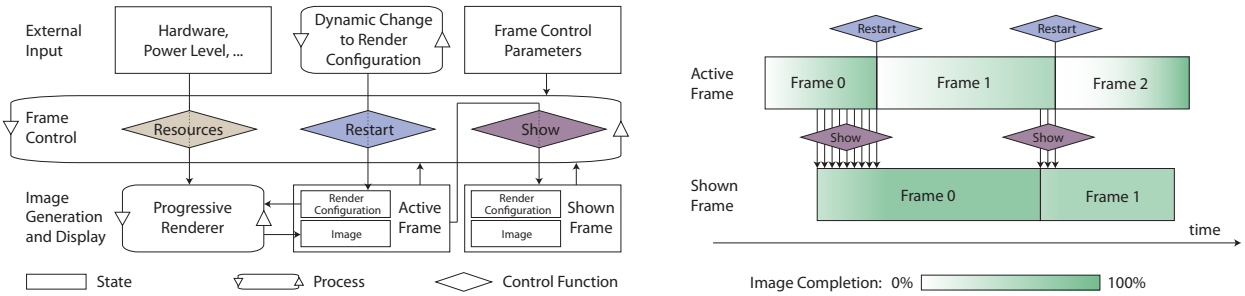
Rendering systems commonly fix either image quality or frame rate during user interaction [2]. For real-time rendering, Wong and Wang [35] model the image generation process by an open-loop approach underpinned by constraints and estimations of its constituents with the goal to achieve as constant frame rates as possible. Their heavy-weight approach describes different rendering processes in detail with all their complexities involved to gain a nonlinear model to relate inputs and outputs using neural networks and fuzzy models. In contrast, Woolley et al. [36] use simple metrics based on image-space distances to steer progressive raytracing. We also take a light-weight approach, requiring only minimal knowledge about the underlying visualization approach, and only requiring minimal assumptions or predictions. In addition, we further concern ourselves with the question of what actually describes an acceptable performance range by means of a perceptual metric as well as a study with visualization researchers.

3 Progressive Visualization Model

Our technique is based on an idealized model of interactive progressive visualization. We derived it from a simple standard workstation setup with a single display. Its extension to setups with more than one camera like stereo rendering, more than one display like multiprojector configurations, or limited display throughput like in remote rendering, is beyond the scope of this paper and remains for future work.

Our model (Fig. 2a) consists of three basic processes: *dynamic change*, *progressive renderer*, and *frame control*. *Dynamic change* comprises the factors that alter a *render configuration* over time, like user interaction or change due to time-varying data. In this work, we address frame-based progressive visualization by means of a *progressive renderer*. This is by far the most widespread variant, an alternative being frameless rendering techniques [6, 11]. The *progressive renderer* continually refines the *active frame*, which reflects an individual *render configuration*. In this sense, a frame consists of both an (intermediate) rendering result of the *active frame* together with the *render configuration* of the *active frame*.

The model follows the principle of double buffering (Fig. 2b). In progressive visualization, a rendering result of the *active frame* is typically already shown while the *active frame* is further refined in the background. In our model, this is triggered by the *show* control function, which copies the *active frame* to the *shown frame* including the respective *render configuration*. If the *active frame* changes through an issued *restart* control function, i.e., the *active frame* is supplied with a new *render configuration*, the *progressive renderer* immediately starts to render a new image of the *active frame* from scratch. How fast



(a) Generic model of interactive progressive visualization.

(b) Exemplified process of double-buffered progressive visualization.

Fig. 2: (a) Progressive visualization model and (b) an illustrating example. *Restart* triggers when to stop the refinement of the *active frame* (back buffer) and start computing a new one instead with the current *render configuration*. *Show* determines when to copy the *active frame* to the *shown frame* (front buffer) for display. *Resources* controls the share of the compute capacity that is consumed by the *progressive renderer*.

the refinement advances can vary significantly, both depending on the *render configuration* (e.g., data set or camera position) as well as the compute resources allotted by the *resources* control function.

In total, our model requires three basic decisions: when to *restart* an *active frame*, when to *show* it, and what share of *resources* to consume in the *progressive renderer*. These three decisions are managed by *frame control*, using the available information about the (previous states of the) interactive system, and based on the *frame control parameters*. In these terms, (traditional) fixed-quality settings only consider the quality of the rendered image of the *active frame*, while (traditional) fixed-frame-rate settings take only into account the time stamp of the last rendered image. As long as there are no changes to the *render configuration*, i.e., *dynamic change* is idle, the image is progressively refined beyond the quality or frame rate limits.

4 Error-Based Frame Control

The fixed limits in typical progressive visualization systems cannot adapt flexibly to *dynamic change* due to camera rotations, transfer function changes, progressing time-dependent data, etc. As exemplified in Fig. 1, this can lead to significant shortcomings in compute-intense interactive visualization sessions. In this section, we introduce a new dynamic *frame control* approach, based on sampling error and response delay, to overcome these issues.

4.1 Motivation and Overview

The main goal of our error-based *frame control* is to minimize the perceived error in the *shown frame* over the course of an interactive visualization session. The error may additionally be balanced against the utilization of resources, which, however, heavily depends on the hardware-related setting, involving several questions. For instance, are the resources shared with other processes or users in the context of a remote visualization server? How are priorities determined? Is power consumption above a certain threshold problematic in the context of mobile devices? Under which circumstances may this threshold be exceeded temporally? While our progressive visualization model can cover these aspects, we use a simplified resource model in our exemplary implementation—a detailed consideration of these questions for different scenarios has to remain for future work.

In this paper, we focus on optimizing *frame control* by minimizing the error during interaction. Hence, offline optimization, i.e., a posteriori error evaluation by means of user studies or video quality metrics [30, 3, 31] is not practicable. Instead, error minimization by *frame control* has to take place online, during user interaction, and introduce very low overhead to avoid delay. This means that neither costly quality evaluation algorithms nor constant manual user adjustment are affordable during interactive visualization.

As a consequence, we split the quality evaluation into an online component (*spatial error estimator* and *temporal error estimator*) and an offline component (*frame control parameters* optimization). Akin to the notion of spatial and temporal errors from video encoders and quality metrics (e.g., [30]), our online component consists of a *spatial error estimator* and a *temporal error estimator* (Sec. 4.2), both providing error estimates at very low overhead. These estimates are then used by our heuristics to operate *restart*, *show*, and *resources* (Sec. 4.3). The heuristics itself is steered by the *frame control parameters*, which are determined and optimized by means of user adjustment, video quality analysis algorithms (Sec. 4.4), or user studies.

4.2 Space-Time Error Estimation

In an interactive visualization system, the temporal error τ shall reflect delayed response to changes, while the spatial error ς reflects compromises in image quality. Naturally, ς and τ are subject to a trade-off: while ς is typically monotonically decreasing with the time spent for rendering, τ is monotonically increasing. This partitioning into two types of error is a good fit for the frame generation pipeline in interactive visualization. The easiest way to implement the *spatial error estimator* would be to directly use the sampling density (e.g., [36]). However, we take a more expressive, yet slightly more expensive content-aware approach that considers color variance (Sec. 5.2). The *temporal error estimator* can be derived directly from the difference between subsequent images, or indirectly by measures operating on subsequent render configurations. For instance, a simple indirect approach could project a precomputed set of vertices surrounding a geometric model to the screen, and then determine the largest difference to the previous projection [36]. However, among other issues, this approach would ignore the degree of detail in the data. In contrast, we use a direct approach and assess the error analogously to the *spatial error estimator* directly from the images (Sec. 5.2). Inherently, this considers the spatial complexity of the data set and allows, e.g., to account for render configurations that are rich in detail by lowering the frame rate.

4.3 Control Heuristics

The temporal and spatial errors determined by the error estimators are used to control *restart*, *show*, and *resources*. For this, we employ the following heuristics. *Restart* strives to achieve the best-possible result for the current *active frame* with respect to errors in time τ and errors in space ς. This is based on the *active frame* and does not consider the *shown frame*. For *show*, both the errors of the current *shown frame* as well as those of the *active frame* are taken into account. The rationale behind this is that the *frame control parameters* are used to determine which combination of temporal and spatial error gives the lower overall error. For *resources*, as good settings highly depend on the use case, no clear, objective metric exists to balance the power or computation time consumed by the *progressive renderer* against the quality of the output. Here, we utilize the spatial error of the *active frame* to determine a maximum spatial error that is acceptable during interaction. In case the *progressive renderer* is capable of overachieving this value, resource usage can be reduced.

While errors could be related directly (e.g., [36]), more flexible approaches are required to introduce a degree of freedom that allows the consideration of advanced aspects in dynamic frame control, like

user preferences (Sec. 6.3). At the same time, parameters should be intuitively adjustable for a user and enable efficient automatic optimization. To achieve these goals, we introduce the *frame control parameters* consisting of a single parameter for each *restart*, *show*, and *resources*, denoted as $\rho, \vartheta, \chi \in [0,1]$, respectively. These parameters are defined by the user or by the automatic optimization process. The *frame control* can be represented by a function ϕ as follows, with multiple actions possible (note that *restart* automatically triggers *show* in our heuristics):

$$\phi = \begin{cases} \textit{restart}, & \text{if } \mu(\rho) \cdot \tau_t^a - \varsigma_t^a \geq 0 \\ \textit{show}, & \text{if } \mu(\vartheta) \cdot (\tau_t^a - \tau_t^s) + (\varsigma_t^a - \varsigma_t^s) \geq 0 \\ \textit{resources} & \text{if } \chi > \varsigma_t^a \vee \delta_t^\chi > \hat{\delta}^\chi \end{cases}, \qquad (1)$$

where ς_t^a and ς_t^s denote the spatial error of the active and the shown frame at time t, respectively, τ_t^a and τ_t^s the temporal error of the active and the shown frame, and $\mu(\cdot)$ maps the parameters from their original range $[0,1]$ to $[0,\infty)$: $\mu(s) = \tan(s \cdot \pi/2)$, δ_t^χ gives the idle time of the resource at time t, and $\hat{\delta}^\chi$ denotes the respective threshold. The rationale behind the triggering of *restart*, *show*, and *resources* is discussed in detail next.

Restart. The temporal error τ_t^a of the *active frame* is weighted with ρ against the spatial error ς_t^a, thus defining the desired trade-off between these two quantities. Note that τ_t^a continuously increases with t, while ς_t^a is continuously decreasing.

Show. Similar to *restart*, we weight temporal against spatial error, but this time with respect to the differences between the *active frame* and the *shown frame*. Practically, the goal is to determine when the decrease in temporal error compensates for the higher spatial error. For all possible settings of ϑ, the active frame is shown as soon as $\varsigma_t^a > \varsigma_t^s$, because $(\tau_t^a - \tau_t^s)$ should always be nonnegative for reasonable implementations.

Resources. In our simple implementation, the resource usage is binary, i.e., either we are progressively refining a frame or idling. Here, the parameter χ defines a static target error value. If the spatial error estimation ς_t^a falls below χ during interaction, we pause rendering until the frame is restarted. We also limit the pause time δ_t^χ to a maximum value $\hat{\delta}^\chi$ (we used $\hat{\delta}^\chi = 1$ s), to ensure that the full (spatial) sampling rate is achieved eventually.

For the purpose of comparison, we also implement an alternative frame control $\overline{\phi}$ for the commonly used fixed image quality $\overline{\omega}$ settings and fixed render time per frame $\overline{\delta}$ settings:

$$\overline{\phi} = \begin{cases} \textit{restart}, & \text{if } (\omega_t^a \geq \overline{\omega} \vee \overline{\delta} \geq \delta_t^a) \wedge \tau^a > 0 \\ \textit{show}, & \text{if } (\omega_t^a \geq \overline{\omega} \vee \overline{\delta} \geq \delta_t^a) \wedge \tau^a = 0 \end{cases}, \qquad (2)$$

where ω_t^a gives the current sampling rate, and δ_t^a denotes the time since the rendering of the *active frame* has started. Practically, this means that for fixed settings, the *progressive renderer* proceeds until the target value is reached, and the *shown frame* is iteratively updated toward the full quality beyond these restrictions only when the *render configuration* is not modified (i.e., the temporal error τ^a is zero).

4.4 Parameter Optimization

The goal of the parameter optimization is to determine *frame control parameters* such that *frame control* delivers the best result according to a metric or user assessment. We distinguish between the following two basic variants: user evaluation and automatic evaluation by means of video quality metrics. In the former, a user simply assesses how well settings for ρ, ϑ, and χ are suited for the scenarios he works on (i.e., a data set, a task to accomplish, etc.) during an interactive session, and chooses the parameters accordingly. In the latter, automatic evaluation makes use of interaction logs recorded during previous interactive visualization sessions. Each interaction log defines a *render configuration sequence*, that along with different hardware settings define *scenarios*. Each *scenario* is rendered with a range of different *frame control parameters* and evaluated by means of a video quality metric. The output of the *quality evaluation* finally provides the basis for

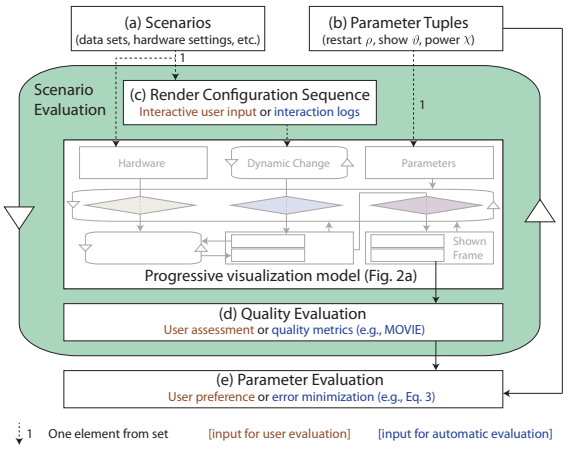


Fig. 3: Our optimization scheme for *frame control parameters* with different variants for user-based (red) and automatic evaluation (blue).

parameter evaluation to determine the best setting over all scenarios (Fig. 3). The respective components are implemented as follows.

(a) *Scenarios.* In our case, each scenario features a specific data set that is explored by means of a sequence of render configurations. We consider the exploration of different volumes in the following. A *scenario* may include computationally weaker systems, which might be simulated by virtually slowing down the test machine through different hardware settings. Optimally, *scenarios* should be representative in terms of the data sets, camera positioning, and so on, with respect to the target field of application.

(b) *Parameter Tuples.* Numerous parameter setting tuples (ρ, ϑ, χ) are considered for evaluation. Previous experience with the system or explicit pre-evaluation can help narrow down the considered range of parameters, thus pruning clearly undesirable settings early for a more efficient evaluation.

(c) *Render Configuration Sequence.* In the course of a user evaluation, the user may simply interact with the respective tool. For the automatic evaluation of our implementation, we use timelined series of changes to the camera setup and the transfer function. In our case, they come from recorded user interaction logs from previous sessions.

(d) *Quality Evaluation.* When a user is evaluating the interactive application, he or she may explicitly provide a score for the experience. Such a score could be determined automatically by measuring the user's performance for predefined tasks. In contrast, in our evaluation in this paper, we do not assess such explicit performance scores, but instead let the user provide his choice of parameters directly (Sec. 6.3). For automatic evaluation, algorithmic video metrics have been used for many years to assess the perceived quality of a video without the need for a user study. We use MOVIE [30] for the evaluation of interactive visualization, due to its high quality results and the availability of the source code. This gives us the possibility to seamlessly integrate it within our parameter optimization pipeline in a distributed environment. MOVIE is a full reference metric that compares a candidate video against a reference, and delivers a single error value as result. We generate these candidate videos by capturing the image from the display buffer 30 times per second. After completing a render configuration sequence for a certain parameter setting, we write the respective image files to disk, convert them to the required YUV video format using FFmpeg, and analyze it with the metric. The whole process runs automatically for given *parameter tuples* and *scenarios*. It also generates a script that can be used directly as input to the grid engine of our compute cluster. This script file defines a job array that invokes as many parallel instances of MOVIE as there are videos that need to be evaluated. As output, among others, MOVIE writes a text file containing a single overall scalar error value for each test, which we use for *parameter evaluation* in the following. Although these video metrics have been extensively researched and evaluated over the past 20 years, they can only be an approximation to the ground truth,

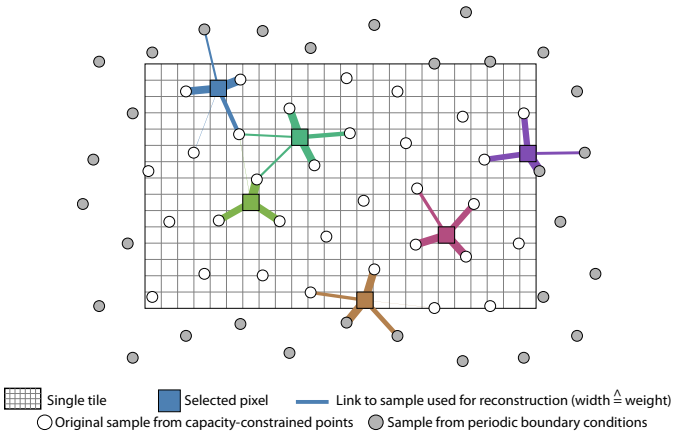


Fig. 4: Reconstruction of pixels from samples and their weight according to a Gaussian filter kernel (Eq. 4), illustrated for colored pixels. Periodic boundary conditions of the sample distribution are explicitly considered. In our implementation, sample weights are precomputed for each tile resolution and stored in sampling table S.

the quality assessment by a human user. A detailed discussion of this aspect can be found at the end of Sec. 6.3.

(e) *Parameter Evaluation*. As indicated above, in the case of manual user evaluation, we simply let the user choose his favorite setting according to his interactive experience (Sec. 6.3). For the automatic evaluation, the error values are determined by the video quality metric for the *m scenarios* and the *n parameter tuples* to find the most favorable setting. This is accomplished by minimizing the least-squares relative error ε on the basis of the error measure $\gamma_\cdot^a(\cdot,\cdot,\cdot)$ from the video metric:

$$\varepsilon = \min_{0<i\leq n} \left(\sum_{j=0}^{m-1} \left(\gamma_j^a(\rho_i,\vartheta_i,\chi_i)/\gamma_j^a(\rho_{b(m)},\vartheta_{b(m)},\chi_{b(m)}) - 1 \right)^2 \right), \tag{3}$$

with $b(m) \in \{0,\ldots,n\}$ denoting the index of the parameter setting giving the lowest error for scenario m. We use relative errors to account for the fact that the absolute error given by a metric can vary significantly (e.g., with the length of an image sequence). In particular, this allows us to avoid any bias toward scenarios with higher error values overall, and to support the combination of different error metrics.

5 Error-Based Progressive Volume Visualization

Despite significant improvements in algorithms and hardware, volume rendering is still computationally challenging for the data sets produced by up-to-date scanners and simulations. Thus, in this section, we discuss the implementation of a progressive, multiresolution GPU volume raycaster for illustrating and evaluating our error-based frame control (Sec. 5.1). In particular, we provide an efficient integrated implementation for the spatio-temporal error estimates (Sec. 5.2).

5.1 Progressive Volume Visualization

The main goals of our progressive volume rendering scheme are performance, flexibility, and simplicity, both with respect to implementation and integration with existing renderers. By flexibility, we mean both the capability to interrupt the renderer at virtually any time, and the absence of any kind of preprocessing or assumptions of coherence across frames. We utilize a multiresolution volume raycaster with optimized sample distribution in image space [4].

Sampling. The renderer uses multiple resolution levels in image space, each of which is subdivided into tiles. The goal is to achieve both flexible interruptability of the *progressive renderer* as well as efficient usage of the hardware. Thus, the granularity (i.e., the tile size) needs to be chosen according to device characteristics. For our CUDA implementation, we determined by experiment that 16K samples per tile are sufficient to efficiently utilize the 512 stream processors of a NVIDIA GTX580 without introducing significant overhead. For the chosen aspect ratio, we generate sample points by means of capacity-constrained point distributions with periodic boundary conditions [4] (Fig. 4). Next, following a quad-tree approach, we subdivide the image space into tiles of increasing resolution levels until there is more than one sample point per pixel. For each resolution level, starting from our original set of samples, we add the required periodic copies to the list of samples. In our experiments, we use WXGA+ (1440×900) as the screen resolution for our renderer, because this constitutes a good trade-off between screen space and the induced cost for automatic video analysis. In total, this results in 5 resolution levels and 341 tiles in total (Fig. 5). In this paper, we measure the *sampling rate* by means of the ratio of completed to total tiles. Akin to the image-space sampling, the sampling distance along rays in object space is doubled, starting from the highest, most detailed resolution level. To render a frame, the tiles are processed from the lowest to highest resolution level. Within each resolution level, they are ordered from the screen center to the outside. Error estimation and *frame control* are carried out between the computation of tiles. The achieved high, sub-frame granularity is essential to be able to react quickly to sudden changes. Our volume raycaster uses a simple local lighting model, making use of gradients that are determined on-the-fly using central differences. The raycaster further makes use of early ray termination, and lighting computations are only executed in non-empty space.

Image Reconstruction and Blending. Pixel color values are reconstructed from the sample points using a Gaussian filter kernel

$$f(d) = \mathrm{e}^{-\alpha d^2} - \mathrm{e}^{-\alpha r^2}, \tag{4}$$

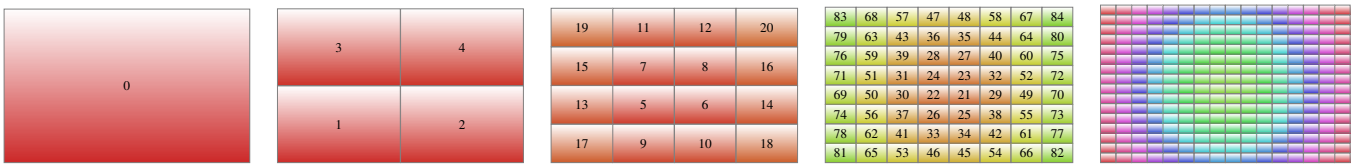
with d denoting the distance in image space, α controlling the falloff, and the radius r ensuring that the filter goes to zero at the boundary of its support [23]. On this basis, in a precomputation step, we generate the sampling table S, that lists for each pixel (x,y) of a tile of sampling resolution l which samples are required for its reconstruction along with their respective weights (Fig. 4). S is then used at runtime in a CUDA kernel to compute the color value of each pixel of a tile (Alg. 1). Finally, for display, we use OpenGL to blend the boundary between tiles of different levels to reduce visual disturbances occurring from visible edges. This is implemented by employing precomputed opacity maps which are generated with respect to the distance of the pixels to the border between tiles of different resolution levels. Results for different sampling rates are shown in Fig. 6.

5.2 Error Estimation

As the basis for *frame control*, fast yet expressive metrics are required for the spatial and temporal error estimators to provide updated values after the rendering of each tile.

Spatial Error Estimation. For each pixel, the estimation of spatial error ς can be efficiently achieved along with image reconstruction such that it only induces marginal computational and no memory lookup overhead (Alg. 1, green). In our implementation, the spatial error is determined by the magnitude of the weighted RGB variance vector over the considered RGB sample color values $\vec{C}(\cdot)$ (Alg. 1, line 16). The underlying incremental algorithm to compute the weighted variance is due to West [34]. Pixel error values are then summed up using a hierarchical reduction scheme on the GPU that makes use of shared memory. Finally, the average pixel error value serves as spatial error estimation. Note that after the completion of each tile only a partial update to the previous error value is required, as only the image region covered by the respective tile needs to be considered. In our implementation, for each pixel covered by the tile, the difference between the previous and the current error value is added to the total value. When *restart* is issued, the spatial error of the active frame ς^a is simply carried over to the shown frame error value ς^s.

Temporal Error Estimation. For temporal error estimation, we first generate a quick approximation of the frame with the current *render configuration*, i.e., the latest camera position, transfer function, and data time step. In detail, we omit lighting, and sample the volume only sparsely during raycasting by using the lowest resolution



Level 0: 1440×900 (1×1), sampling rate 0.0–0.3%

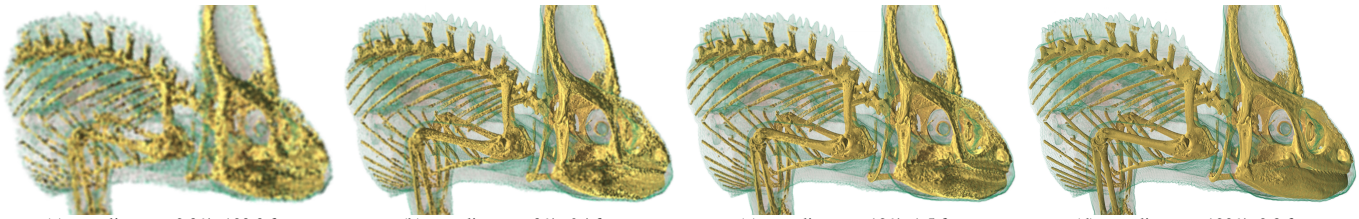
Level 1: 720×450 (2×2), sampling rate 0.3–1.5%

Level 2: 360×225 (4×4), sampling rate 1.5–6.2%

Level 3: 180×113 (8×8), sampling rate 6.2–24.9%

Level 4: 90×57 (16×16), sampling rate 24.9–100%

Fig. 5: Tiles at different resolution levels generated for a 1440×900 screen. Their order of processing is depicted by numbers and colors.



(a) sampling rate 0.3%, 103.3 fps (b) sampling rate 3%, 6.4 fps (c) sampling rate 10%, 1.5 fps (d) sampling rate 100%, 0.2 fps

Fig. 6: Different sampling rate settings and achieved frame rates with our renderer for the Chameleon data set.

Algorithm 1 Integrated image reconstruction and error estimation. Spatial error estimation implementation is based on incremental variance computation and depicted in blue. Temporal error is simply computed from the difference to the previous color as shown in green.

1: **procedure** RECONSTRUCTION(x,y,l) ▷ pixel coordinates (x,y) in tile with resolution level l
2: $w_\Sigma, \vec{c}_\Sigma \leftarrow 0$ ▷ initialize sum of weights and sum of colors
3: $\vec{m}, \vec{m}_2 \leftarrow 0$ ▷ initialize mean and squared differences from mean
4: **for all** $(i,w) \in S(x,y,l)$ **do** ▷ fetch sample index i and weight w from sampling table S
5: $\vec{c}_t \leftarrow \vec{C}(i)$ ▷ lookup sample color from output of renderer
6: $\vec{c}_\Sigma \leftarrow \vec{c}_\Sigma + w \cdot \vec{c}_t$ ▷ update weighted color sum
7: $w'_\Sigma \leftarrow w_\Sigma$ ▷ back up sum of weights from previous iteration
8: $w_\Sigma \leftarrow w_\Sigma + w$ ▷ update sum of weights
9: $\vec{d} \leftarrow \vec{c}_t - \vec{m}$ ▷ deviation from current mean
10: $\vec{d}_w \leftarrow \vec{d} \cdot \frac{w}{w_\Sigma}$ ▷ relative weighted deviation from current mean
11: $\vec{m} \leftarrow \vec{m} + \vec{d}_w$ ▷ update mean
12: $\vec{m}_2 \leftarrow \vec{m}_2 + w'_\Sigma \cdot \vec{d} \cdot \vec{d}_w^T$ ▷ update sum of squared differences from current mean
13: **end for**
14: $\vec{c} \leftarrow \vec{c}_\Sigma / w_\Sigma$ ▷ output color for pixel (x,y)
15: $\vec{\sigma}^2 \leftarrow (\vec{m}_2 / w_\Sigma) \cdot |S(x,y)| / (|S(x,y)| - 1)$ ▷ compute variance
16: $\varsigma \leftarrow |\vec{\sigma}^2|$ ▷ spatial error ς is length of variance vector
17: $\tau' \leftarrow |\vec{c} - \vec{c}_{\text{prev}}|$ ▷ τ' is difference between $\vec{c}$ and prior color $\vec{c}_{\text{prev}}$
18: **return** $(\vec{c}, \varsigma)$ **or** $(\vec{c}, \tau')$ ▷ resulting pixel and respective error
19: **end procedure**

tile and significantly increase the step size along rays (we use a factor of 50 with respect to the highest level). The respective temporal error is computed by means of per-pixel color differences of the current to the previous approximate rendering (Alg. 1, line 17). Next, the temporal error values τ' of all pixels are summed up by employing the hierarchical scheme that is also used for spatial error estimation. This value is added to the total temporal error value τ^a of the active frame and the shown frame τ^s, with $\tau^a = 0$ after each *restart*. As with the spatial error value, the temporal error value of the active frame τ^a is carried over to the error of the shown frame τ^s in case of a *show* or *restart*. Note that in contrast to the spatial error that is computed during the reconstruction of every tile for display, the temporal error is only evaluated in case of changes to the *render configuration* since the last assessment.

6 Application and Evaluation

For running our measurements, we use a NVIDIA GTX580 and an Intel Core i7 with 3.4 GHz. We employ four data sets from both CT scans and simulations in our evaluation (references to renderings and data set resolution are given in brackets): the Chameleon data set (Fig. 6, $1024 \times 1024 \times 1080$), the Jet data set (Figs. 8(a) and (b), $720 \times 320 \times 320$), the Flower data set (Fig. 1, $1024 \times 1024 \times 1024$), and the time-dependent Vortex data set (Figs. 8(c)–(e), $529 \times 529 \times 529$, with 60 time steps). All compute-intense steps required for interactive rendering are executed in parallel on the GPU using CUDA, particularly including the volume raycaster and the reconstruction along with the error estimation. For our 1440×900 screen resolution, the image reconstruction takes 5 ms for the lowest level, 1.5 ms for the second-lowest level, and below half a millisecond for all subsequent levels. This decrease in reconstruction time exhibits a roughly linear dependence on the decreasing number of pixels covered by a tile (Fig. 5). To reduce this cost for higher levels, a hybrid reconstruction and upscaling approach could be investigated for future work.

As discussed in Sec. 5.2, computing the spatial error through the variance of the samples can be done without significant overhead, basically requiring only a couple of additional floating point operations. Determining the temporal error basically consists of two steps, namely the approximate rendering and the integrated reconstruction and difference computation to the previous approximation. In our measurements, the approximate sampling was in the range of 1% to maximally 10% of the full render time of a tile, while the subsequent reconstruction and error computation always took around 0.5 ms (e.g., for reference, the sampling of tile 22 in Fig. 6 took 24 ms). This means that the computational overhead for assessing the temporal error is fairly low, particularly when considering that this is only done in case of changes to the *render configuration*.

In the following, we first demonstrate the adaptive behavior of error-based frame control by means of a specific example (Sec. 6.1). Second, we perform automatic parameter optimization and evaluation using the MOVIE video quality metric (Sec. 6.2). Third, we conduct an expert study with visualization researchers to optimize the parameter settings, evaluate the utility of our approach against alternatives, and compare the results to automatic evaluation (Sec. 6.3).

6.1 Practical Example of Error-Based Frame Control

We showcase the characteristics of error-based frame control at the example of interactive exploration of the Chameleon data set (Fig. 7). In the first 100 frames the data set is quickly rotated. This induces a high temporal error and accordingly frame rates around 25 fps for more flu-

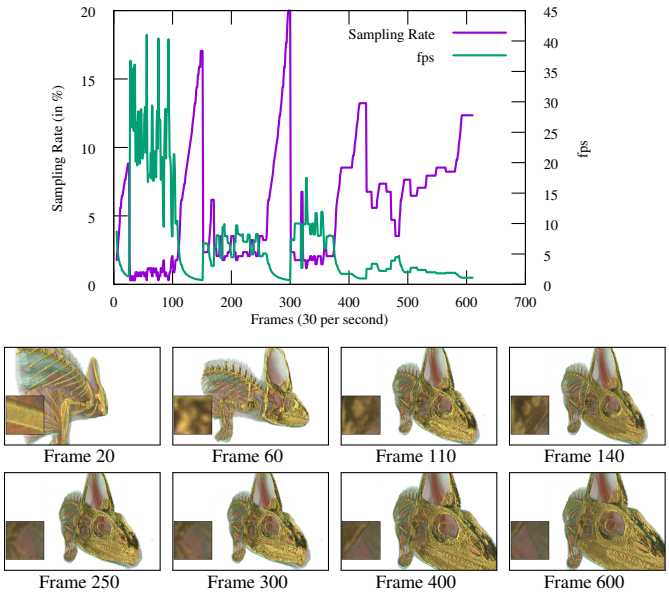


Fig. 7: Camera path for the Chameleon data set. *Top:* Sampling rate and frame rate for $\rho = 0.6$. *Bottom:* Renderings for selected frames with close-up.

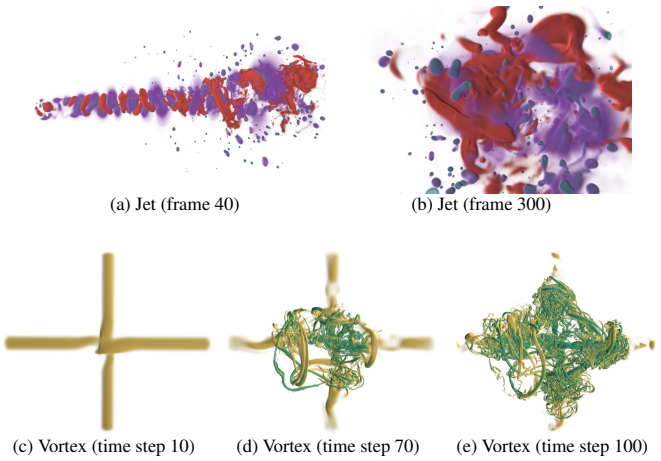


Fig. 8: Key frames of the videos from the simulation data sets that were used for automatic evaluation. In the respective camera path, the Jet data set was quickly rotated from the side view (a) to the tip of the pressure advancement (b), which was then investigated in detail. The Vortex series (c)–(e) shows the temporal development of the vortex cascade, visualized with the λ_2 criterion [17].

ent navigation. From frame 110 to 140, no changes occur for almost a second, which leads to a continuous refinement of the current frame, as can also be seen from the increasing sampling rate. Then, there is a sequence of slower camera movement (frames 150–250), followed by another phase of no change (frames 250–300). Then, after slower changes to the camera position (frames 300–380), a certain feature is investigated in detail. This requires a high sampling rate to enable the user to assess fine details. We refer the readers to the accompanying video for further details.

6.2 Automatic Evaluation Using a Video Quality Metric

By means of automatic evaluation, we aim to optimize the parameters ρ for *restart* and ϑ for *show*, and study the impact of the power reduction parameter χ. We considered seven scenarios in total: the Chameleon, Jet, and Flower data sets, each with one sequence adjusting the camera and one adjusting the transfer function, and the Vortex data set receiving a new time step every 100 ms. As input for MOVIE, we compute a video for each parameter setting along with a reference video that batch renders a full-quality image for every frame. The videos are generated from transformations that were recorded from user interaction. To be representative for a variety of user actions, they contain both slow and fast-paced changes. These videos are only between six and twenty seconds long to keep computation time both for generation and subsequent evaluation within reasonable limits.

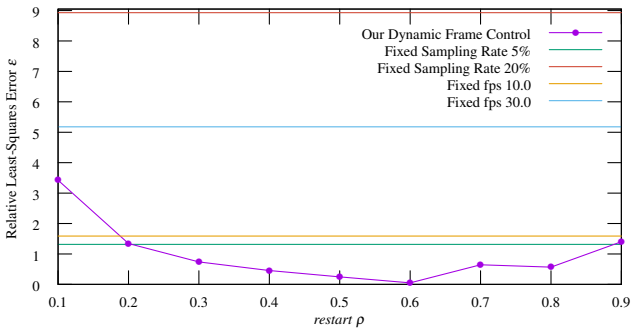
In our experiments, the evaluation of a video with a resolution of 1440×900 and 30 frames per second using MOVIE took twelve to sixteen hours on an Intel Xeon processor running at 2.4 GHz. By using a small cluster, we could process up to 64 videos in parallel. However, considering variations of multiple parameter values in one measurement series would still take a prohibitively long time. Thus, we use a multi-stage process to significantly cut down the number of evaluations. First, we optimize the restart parameter ρ, which has been shown by our previous experiments to be of predominant impact in comparison to the show parameter ϑ. Then, for the resulting optimal setting for ρ, we evaluate different parameter settings for ϑ. On this basis, we finally examine the impact of the resource parameter χ.

For the restart parameter ρ, we consider values from 0.1 to 0.9 in steps of 0.1. We further include static frame rates with 10 fps and 30 fps as well as static sampling rates of 5% and 20% in our evaluation. Fig. 9a shows the resulting least-squares relative error over all scenarios (as discussed in Sec. 4.4). According to this, error-based frame control with $\rho = 0.6$ achieves the smallest error value overall. The relatively high static sampling rate (20%) that delivers high-quality renderings but frame rates below 1 fps delivers the worst result. Here, low quality settings (5%) favoring higher frame rates improve the results significantly. For static frame rates, 10 fps delivers significantly lower errors than 30 fps, meaning that it provides a better trade-off between sampling rate and responsiveness for the considered cases.

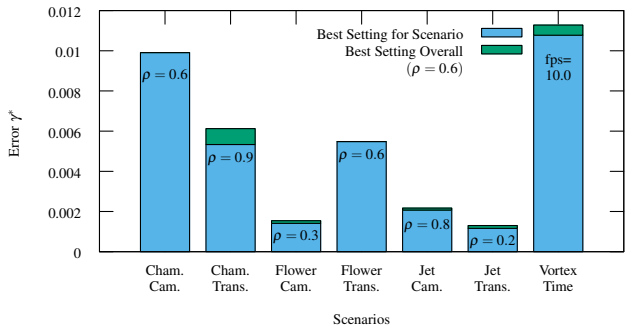
As indicated by the low slope of error values for ρ across a wide range of values in Fig. 9a, a variety of different settings are potential candidates for delivering the best result in one specific scenario (Fig. 9b). However, it can be seen that the determined optimal setting $\rho = 0.6$ is only marginally worse in any scenario, with respect to the best result for each scenario individually. Our error-based control further yields the lowest error for each scenario individually, except for the Vortex series. Naturally, its discrete refreshes 10 times a second yields good results with the 10 fps setting. In total, it can be seen that error-based frame control with $\rho = 0.6$ can be used successfully across all considered use cases without requiring further adjustment.

Based on $\rho = 0.6$, we then investigate the impact of the show parameter for the same scenarios with the following settings: $\vartheta \in \{0, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.5\}$. Our results indicate that the lowest overall error is achieved with $\vartheta = 0$. This means that *show* is only issued when a frame is restarted or when the spatial error of the active frame is smaller than that of the shown frame. While a more early show operation can reduce the visible temporal error and allow the frame to still refine further, we think this result is due to the induced strong variation of displayed image quality, leading to significant flickering.

Finally, Fig. 10 shows the impact of the power parameter χ on the resource usage and the error determined by MOVIE. In our implementation, χ defines an error threshold with respect to our spatial error estimation. Most prominently, it can be seen that a quite significant reduction of resource usage can be achieved with only little increase in error. This is due to the fact that an increasing sample count has decreasing impact on the perceived image quality, i.e., the resource usage can be lowered for sampling rates beyond a certain threshold with merely sub-linear quality impact. The negative effect of trading resource usage against sampling rate is further significantly dampened by the adaptivity of our approach, i.e., resource utilization is only reduced in adequate situations as defined by the user. More elaborate schemes and a more detailed evaluation of its effects, e.g., on explicit power consumption, remain for future work.



(a) Least-squares relative errors over all considered scenarios for different values for ρ.



(b) Error of the most optimal restart parameter setting for each scenario, plotted against the most optimal global setting of $\rho = 0.6$ from (a).

Fig. 9: Automatic optimization of the restart parameter ρ using the MOVIE video metric and a variety of different scenarios.

6.3 Expert Review

Scope and Structure. The primary goal of our expert review was to evaluate the usefulness of our approach for volume rendering against fixed settings for frame rate and quality (denoted as *modes* below). Further objectives of the study were to identify preferred parameter settings, and to assess similarities and differences to the automatic evaluation from Sec. 6.2. Five visualization researchers evaluated our implementation by interactively exploring the Chameleon and the Jet data set. They are primarily concerned with the development of new visualization techniques, but also use their own and other interactive tools for analysis on a regular basis. Each participant spent 45 minutes to 1 hour interacting with our implementation.

The procedure was loosely structured into three phases. First, we asked the participants to make themselves familiar with the volume rendering tool and the different modes of the program. Second, we gave them a number of exploration tasks to accomplish and asked them to evaluate the usefulness of our tool along the way. In this phase, the parameter values of the different modes were fixed as follows: the static frame rate was set to 30 fps, while the sampling rate was at 20%, and the restart parameter ρ was set to 0.6. For simplicity and time constraints, the show and power parameters were set to 0 and ignored throughout the study. A task consisted either of matching a certain camera view or transfer function to a precomputed visualization result. Twenty randomly distributed and uniformly colored marker crosses were added to the volume to support the matching. Tasks were structured into groups of three (one for each mode), and the order in which the modes had to be used was shuffled quasi-randomly. Finally, there was a free exploration phase in which we asked the participants to explore different parameter settings, and compare the different modes freely to determine the one they like best. After phase two, the participants were handed a small questionnaire. They were asked to complete the part about their experience so far right away, and fill in the remaining parts during the final phase three. The questionnaire contained both multiple choice fields regarding the preferred modes as well as text fields for providing comments regarding different aspects. In brief, we asked them to answer the following questions (answers given to quantitative questions are listed in Table 1):

- Order of suitability of the modes for the assigned tasks for the fixed parameter settings (Table 1, *O* Tasks)?
- Best parameter setting for each mode for the free exploration of the Chameleon (*P* Cham.) and Jet data set (*P* Jet)?
- Remarks regarding the parameter choice?
- Order of suitability of the modes to gain a fast overview of the data set (*O* Overv.), a detailed analysis of certain features (*O* Det.), and overall (*O* Tot.)?
- Justification for the preference ratings?
- Miscellaneous comments regarding this study?

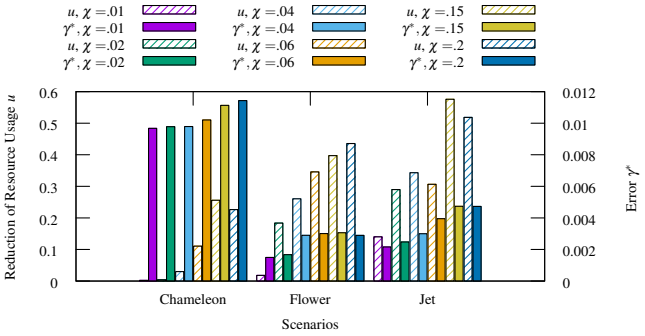


Fig. 10: Impact of power parameter settings χ on resource usage u and error γ^* for the camera path scenarios with $\rho = 0.6$ and $\vartheta = 0$.

User Comments. First, the participants were asked to assess the usefulness of the different modes with respect to the camera configuration and transfer function matching tasks. Overall, all participants found the fast response time of the static frame rate very helpful, particularly for quick camera rotations. However, Participants 2 and 4 noted that this also loses a significant amount of detail in the data set. With respect to the provided static sampling rate, all participants complained that while providing detailed renderings, its significant delays impede precise navigation. Our error-based frame control was rated as being a good compromise between responsiveness and detail (Participant 4), which adaptively allows for a high enough frame rate but also provides detail (Participants 0 and 3). Participants 0 and 1 additionally noted that they found it particularly helpful for matching transfer functions. However, also for transfer function matching, Participant 2 perceived the delays for small changes to be a little too long with our error-based control. In summary, error-based frame control was chosen over static frame rate or quality as the overall preferred method for the task phase. However, there were some remarks that the tasks were not emphasizing the properties and characteristics of the different modes strongly enough. For instance, Participant 0 noted that the disadvantage of low quality for static frame rate did not affect the tasks that much because the provided orientation markers were still visible. For future work, we would like to more closely emulate real world tasks in a more extensive study.

Next, in the exploration phase, the participants were asked to navigate freely, i.e., to explore the data set on their own and determine their personal preferences this way. We summarize their comments in the following. It was noted that the required sampling rate in general strongly depends on the data set and that it is thus hard to set for general purposes (Participants 0, 3, and 4). According to Participant 3, it also always bears the potential of sudden drops in frame rate should the rendering cost change quickly. Thus, the parameter setting highly

depends on what the goal of the exploration is (Participant 0). As a result, Participant 2 found the static sampling rate unpleasant to use, particularly for longer sessions. Most of the time during interactive exploration, the effects of static sampling rate are either choppy movement or low rendering quality that misses important features (Participant 1). While static frame rate delivers better results more independently from the underlying data set (Participant 4), the strong loss of quality that occurs even for only slight changes was found unpleasant, particularly for cases in which details are of importance (Participant 3). Error-based frame control was generally found to be "a good compromise between static frame rate and static sampling rate" (Participant 3). Participant 1 stated that error-based frame control is most appropriate for adjusting transfer functions and the camera position, both when it comes to slow and fast movement. Participant 0 particularly preferred error-based frame control for detailed adjustments of the transfer function because more detailed and thus more helpful renderings were available during interaction.

Ratings and Parameter Settings. Like the participant's comments discussed above, the order of suitability selected by the participants in the questionnaire clearly reflect a preference toward error-based frame control (Table 1). They favored it four to one for detailed investigation of the data set as well as for overall usage, with the other one being static frame rate. While a static frame rate delivering high quality would be great for detailed investigation once an interesting spot has been reached, getting there is cumbersome due to the involved sluggishness. For just getting a quick, rough overview of the data sets, comments and selected preferences suggest that both a high static frame rate and error-based frame control are well suited for this use case. This could be expected, as for faster movement, error-based frame control leads to high fps as well. However, the parameter settings vary in a certain range with specific settings depending on general user preferences toward high quality or responsiveness. For the static sampling rate, parameter settings in the low range between 2% and 7.5% were chosen, despite the significant visual disturbances associated with that (e.g., Fig. 6b). For static frame rate settings, preferred settings vary between 10 fps or 30 fps, with a slight preference overall toward the lower end for higher visual quality. Parameter settings for the restart parameter ρ range between 0.3 and 0.7, with a preference toward the higher end, i.e., toward higher frame rates. Furthermore, preferred settings may vary with the data set, with lower values for the both more expensive to render and complex structured Chameleon in comparison to the Jet data set. Relating to this, Participant 1 stated that the Jet data set contains less detail, and thus different settings seem adequate as compared to the Chameleon data set.

Comparison to Automatic Evaluation. In essence, the results from the user study confirm the basic trend from the automatic evaluation (Sec. 6.2 and Fig. 9). For the static sampling rate, relatively low quality settings are preferred by the users, as they allow for fluid interaction for a wide range of camera and transfer function configurations. In the automatic evaluation, this is reflected by the much lower relative least-squares error ε for the lower sampling rate setting in Fig. 9a. For static frame rate, user preferences range approximately between 10 fps or 30 fps, i.e., varying in its trend toward image quality or responsiveness. Automatic evaluation exhibits basically the same trend, yet a lot more distinctively (Fig. 9a). For error-based frame control, the restart parameter $\rho = 0.6$ was determined as the best setting by the automatic evaluation, with lower values exhibiting only slightly, yet continuously worse results. Similar settings were also popular with the participants.

Regarding the chosen preferences with respect to the mode, both the user study and the automatic evaluation clearly favored our error-based frame control. However, while user preferences lean toward fixed frame rate when compared to fixed sampling rate, they perform about equally well with their optimal parameter setting according to automatic evaluation (Fig. 9a). Note that for the results in Sec. 6.2, four different data sets were used, while only two were part of the expert study here. More definite qualitative and quantitative statements would require an automatic evaluation with a wider range of data sets and performed interactions, a more extensive expert study with more participants, and possibly additional video metrics. In particular, while video metrics have the important advantage of allowing for automatic evaluation and optimization, they are optimized for determining the quality of videos, which might differ from the user experience of an interactive visualization tool. Here, we believe that more research is required to better quantify these differences.

Table 1: Ratings given by the visualization experts in the following order: *fixed frame rate*, *fixed sampling rate*, *error-based frame control*. *O* stands for the order of preference, 1 being the most preferable to 3 being the least preferable. *P* stands for the selection of the best-suited parameter value for each of these. *Tasks* denotes the rating of the task phase of our evaluation. *Overv.* denotes the suitability for getting a quick overview of the data set, while *Det.* stands for the suitability for the detailed analysis of a certain feature. *Tot.* gives the overall rating.

Id	*O* Tasks	*P* Cham.	*P* Jet	*O* Overv.	*O* Det.	*O* Tot.
0	2 3 **1**	5 5 0.2	5 6 0.07	**1** 3 2	2 3 **1**	2 3 **1**
1	**1** 3 2	28 2 0.3	25 2 0.7	**1** 3 2	2 3 **1**	2 3 **1**
2	**1** 3 2	9 2 0.7	9 4 0.6	2 3 **1**	**1** 3 2	**1** 3 2
3	2 3 **1**	10 3 0.3	25 5 0.6	2 3 **1**	2 3 **1**	2 3 **1**
4	2 3 **1**	10 2 0.15	10 7.5 0.2	**1** 3 2	2 3 **1**	2 3 **1**

7 Conclusion

We presented a novel scheme for progressive rendering in interactive visualization that is capable to dynamically adapt to the different situations that occur during data exploration. For this error-based frame control model, we provided an efficient implementation of a volume raycaster, featuring integrated GPU-accelerated image reconstruction and error estimation. Our implementation uniformly handles changes due to camera transforms, transfer function adaptations, as well as the progression of volume data to new time steps. We further demonstrated an automatic parameter optimization scheme using a video metric to optimize our frame control, and finally showed its practical utility by means of an expert study with visualization researchers.

For future work, we plan to significantly expand the aspect of resource utilization and study the possibilities of intelligently decreasing power consumption in more detail. We would further like to extend our evaluation by means of other video metrics (like DRIVQM [3]) and a more extensive user study. This user study should incorporate a more diverse group of users, particularly featuring application domain scientists, and consider the experiences of users utilizing it in their everyday work. In this context, taking the characteristics of human interaction into account more comprehensively could also be a promising direction. We would further like to implement and evaluate our error-based control scheme for an extended raycaster (featuring other data types and out-of-core rendering), as well as other visualization techniques beyond volume rendering. In addition, limiting ourselves to one technique for generating frames, like to raycasting in this paper, cannot avoid significant (temporal and/or spatial) artifacts in cases in which the gap is too large between the cost of this technique and the power of the available compute resources. The extension to hybrid approaches that switch to other, computationally cheaper techniques (like warping [25, 32]) when required could help handling these cases in a more suitable way for the user. Finally, considering additional information like the rate of incoming data (e.g., with simulations running in parallel), or outgoing images over the network in remote rendering could allow for more efficient frame control in such scenarios, too.

Acknowledgements

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

REFERENCES

[1] E. W. Anderson, K. C. Potter, L. E. Matzen, J. F. Shepherd, G. A. Preston, and C. T. Silva. A user study of visualization effectiveness using EEG and cognitive load. *Computer Graphics Forum*, 30(3):791–800, 2011.

[2] Autodesk, Inc. Optimize hardware rendering for frame rate or quality, 2014.

[3] 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*, SIGGRAPH ASIA '10, pages 161:1–161:12, 2010.

[4] M. Balzer, T. Schlömer, and O. Deussen. Capacity-constrained point distributions: A variant of Lloyd's method. *ACM Transactions on Graphics*, 28(3):86:1–86:8, 2009.

[5] 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*, 26(4):399–412, Nov. 2012.

[6] G. Bishop, H. Fuchs, L. McMillan, and E. J. S. Zagier. Frameless rendering: Double buffering considered harmful. SIGGRAPH '94, pages 175–176, 1994.

[7] M. R. Bolin and G. W. Meyer. A perceptually based adaptive sampling algorithm. In *Proceedings of the 25th annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 299–309, 1998.

[8] S. P. Callahan and C. T. Silva. Accelerating unstructured volume rendering with joint bilateral upsampling. *Journal of Graphics, GPU, and Game Tools*, 14(1):1–15, 2009.

[9] K. Claypool and M. Claypool. On frame rate and player performance in first person shooter games. *Springer Multimedia Systems Journal*, 13(1):3–17, 2007.

[10] M. Claypool and K. Claypool. Perspectives, frame rates and resolutions: It's all in the game. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 42–49, 2009.

[11] A. Dayal, C. Woolley, B. Watson, and D. Luebke. Adaptive frameless rendering. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, 2005.

[12] J.-P. Farrugia and B. Péroche. A progressive rendering algorithm using an adaptive perceptually based image metric. *Comput. Graph. Forum*, 23(3):605–614, 2004.

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

[14] S. Huang, S. Xiao, and W. Feng. On the energy efficiency of graphics processing units for scientific computing. In *Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8, 2009.

[15] Q. Huynh-Thu and M. Ghanbari. Temporal aspect of perceived quality in mobile video broadcasting. *IEEE Transactions on Broadcasting*, 54(3):641–651, 2008.

[16] T. Isenberg, P. Isenberg, J. Chen, M. Sedlmair, and T. Moller. A systematic review on the practice of evaluating visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2818–2827, 2013.

[17] J. Jeong and F. Hussain. On the identification of a vortex. *Journal of Fluid Mechanics*, 285:69–94, 1995.

[18] B. Johnsson, P. Ganestam, M. Doggett, and T. Akenine-Möller. Power efficiency for software algorithms running on graphics processors. In *High-Performance Graphics*, pages 67–75, 2012.

[19] A. Kratz, J. Reininghaus, M. Hadwiger, and I. Hotz. Adaptive screen-space sampling for volume ray-casting. Technical Report 11-04, Zuse Institute Berlin, Takustr.7, 14195 Berlin, 2011.

[20] D. Laidlaw, R. Kirby, C. Jackson, J. Davidson, T. Miller, M. da Silva, W. Warren, and M. Tarr. Comparing 2D vector field visualization methods: a user study. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):59–70, 2005.

[21] M. Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, 6(1):2–7, 1990.

[22] R. S. Overbeck, C. Donner, and R. Ramamoorthi. Adaptive wavelet rendering. *Proceedings of ACM SIGGRAPH Asia 2009*, 28(5):1–12, 2009.

[23] M. Pharr and G. Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2nd edition, 2010.

[24] J. Pool, A. Lastra, and M. Singh. Precision selection for energy-efficient pixel shaders. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, pages 159–168, 2011.

[25] H. Qu, M. Wan, J. Qin, and A. Kaufman. Image based rendering with stable frame rates. In *Proceedings of the IEEE Conference on Visualization '00*, pages 251–258, 2000.

[26] M. Ramasubramanian, S. N. Pattanaik, and D. P. Greenberg. A perceptually based physical error metric for realistic image synthesis. SIGGRAPH '99, pages 73–82, 1999.

[27] M. Ribble. Power friendly GPU programming. In *ACM SIGGRAPH 2012 Course: Beyond Programmable Shading*, SIGGRAPH '12, page 29, 2012.

[28] F. Rousselle, C. Knaus, and M. Zwicker. Adaptive sampling and reconstruction using greedy error minimization. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pages 1–12, 2011.

[29] J. Sanyal, S. Zhang, G. Bhattacharya, P. Amburn, and R. Moorhead. A user study to compare four uncertainty visualization methods for 1D and 2D datasets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1209–1218, 2009.

[30] K. Seshadrinathan and A. C. Bovik. Motion tuned spatio-temporal quality assessment of natural videos. *IEEE Transactions on Image Processing*, 19(2):335–350, 2010.

[31] H. Sheikh and A. Bovik. Image information and visual quality. *IEEE Transactions on Image Processing*, 15(2):430–444, 2006.

[32] H.-W. Shen and C. R. Johnson. Differential volume rendering: a fast volume visualization technique for flow animation. In *Proceedings of the conference on Visualization '94*, pages 180–187, 1994.

[33] M. Tory and T. Möller. Human factors in visualization research. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):72–84, 2004.

[34] D. H. D. West. Updating mean and variance estimates: An improved method. *Communications of the ACM*, 22(9):532–535, 1979.

[35] G. Wong and J. Wang. *Real-time rendering: Computer graphics with control engineering*. CRC Press, Boca Raton, FL, 2014.

[36] C. Woolley, D. Luebke, B. Watson, and A. Dayal. Interruptible rendering. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics*, pages 143–151, 2003.