Balanced Sampling and Compression for Remote Visualization

Steffen Frey[*], Filip Sadlo[†], and Thomas Ertl[‡]
*‡: University of Stuttgart, †: Heidelberg University

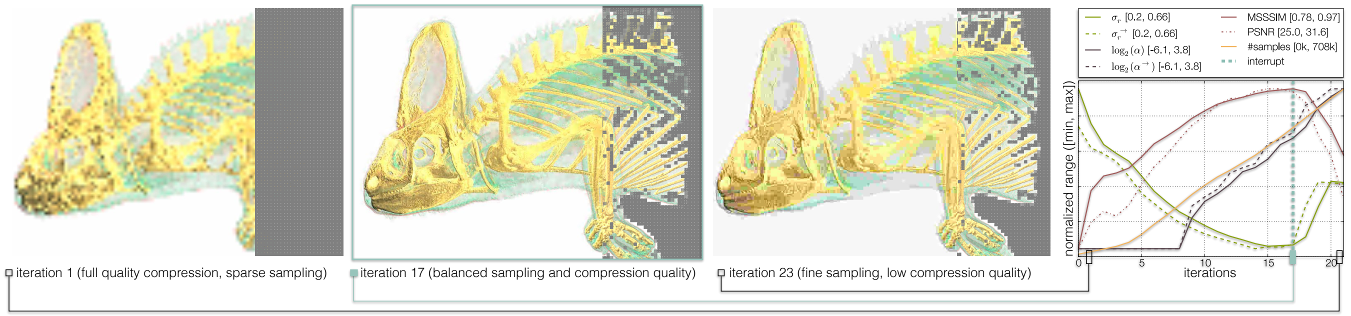


Figure 1: *Our progressive remote visualization scheme adaptively places samples (right quarter of renderings). Sampling (#samples) and compression rate (α) are adjusted such that the requested frame response latency is reached. We dynamically determine the best trade-off by estimating the frame quality (σ and $\sigma^{\rightarrow}$) of the current and next iteration using regression analysis (cf. quality metrics PSNR and MSSSIM).*

Abstract

We present a novel approach for handling sampling and compression in remote visualization in an integrative fashion. As adaptive sampling and compression share the same underlying concepts and criteria, the times spent for visualization and transfer can be balanced directly to optimize the image quality that can be achieved within a prescribed time window. Our dynamic adjustments regarding adaptive sampling, compression, and balancing, employ regression analysis-based error estimation which is carried out individually for each image block of a visualization frame. Our approach is tuned for high parallel efficiency in GPU-based remote visualization. We demonstrate its utility within a prototypical remote volume visualization pipeline by means of different datasets and configurations.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Display Algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Radiosity;

Keywords: remote visualization, adaptive volume rendering

1 Introduction

Remote visualization is an essential component at the intersection of cloud, big data, and high-performance computing. Increasing dataset sizes resulting from increasingly distributed compute environments (e.g., large-scale supercomputer simulations) make it progressively impractical to require data locality for interactive visual analysis. As a consequence, remote desktop software for high-performance evaluation is widely used to check the progress online and obtain live results of long-running simulations (e.g., [Ma and Camp 2000]), and has also been discussed in the context of mobile devices [Diepstraten et al. 2004], and protection of proprietary data [Koller et al. 2004], among others. Typically, the image generation and transfer parts are controlled independently, e.g., by means of adaptive sampling in image space [Levoy 1990; Bolin and Meyer 1995], and by the quality of the lossy compression used for transfer (like for JPEG or MPEG [Koller et al. 2004; Herzog et al. 2008]), respectively. Several approaches targeted toward visualization have been proposed to further individually reduce the time for transfer, e.g., Pajak et al. [2011] discuss efficient compression and streaming, as well as heavy server load, for instance, by means of level-of-detail techniques [Moreland et al. 2008].

However, the time required each for visualization and transfer is influenced by a variety of factors, including the hardware used for rendering, the complexity of the dataset, the viewpoint, the network connection of the visualization server to the client, etc. As a result, the achieved latency varies hugely within and across different scenarios, and different parameter settings need to be chosen explicitly to harmonize image generation and transfer, and yield the optimal result with respect to user-defined constraints.

In this paper, we optimize the remote rendering process toward achieving the best quality for each individual frame with the constraint of a fixed response latency to a user request. We directly relate the losses stemming from undersampling and lossy compression to dynamically balance the time spent for visualization against the time spent for transfer, and optimize the efficiency of the time spent in each individual task. No dataset-specific preprocessing is required, i.e., no additional delay is introduced for visualizing the latest time steps from a running simulation. In brief, we contribute an approach for remote visualization that shares concepts, criteria, and computation steps for an integrated lossy frame compression and adaptive volume raycasting (Sec. 2), that

- adaptively prioritizes regions for refinement with a novel progressive rendering scheme,
- dynamically chooses the quantization quality for compression to meet latency restrictions,
- and determines the best time share for visualization and transfer for each frame on-the-fly (Sec. 3).
- We employ regression analysis-based error prediction considering both undersampling and lossy compression for driving the decision making (Sec. 4).

[*]e-mail:steffen.frey@visus.uni-stuttgart.de
[†]e-mail:filip.sadlo@iwr.uni-heidelberg.de
[‡]e-mail:thomas.ertl@visus.uni-stuttgart.de

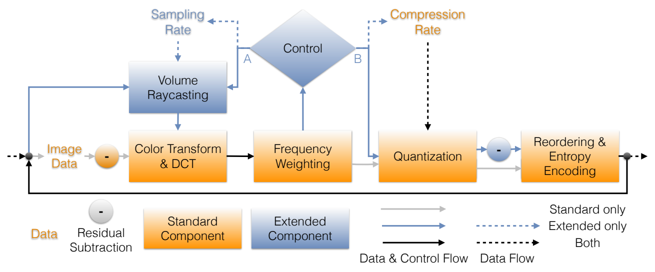


Figure 2: *Standard and extended image-data streaming pipeline.*

2 Remote Visualization Server Architecture

Fig. 2 shows a standard video encoding pipeline (in orange) with our modifications (in blue). In a standard video encoding pipeline, one starts with Image Data that needs to be transferred, and typically some kind of frame prediction is subtracted from the provided image to improve compression efficiency later on (Residual Subtraction). In our remote visualization setup, this image is generated and refined iteratively during the whole process. We do this on the basis of a frame request from the client that typically includes the camera configuration, but may also include adjustment to the transfer function, time step of the dataset, etc. In this work, we employ Volume Raycasting with early ray termination, and a simple local lighting model for which gradients are determined on the fly using central differences. Here, the residual subtraction step is delayed to the end of the pipeline as we require full frame information later on (Residual Subtraction).

Subsequently, the RGB samples are transformed into YCbCr color space and partitioned into blocks, with each block being translated independently into its frequency representation using the discrete cosine transform (Color Transform & DCT). Note that depending on the state of refinement, Volume Raycasting produces sparse image-based sampling, and we use an adjusted DCT transform that can handle these adequately, following the description given by Bolin et al. [1995] (resulting in as many frequency terms as there are samples in a block). Weighted frequencies F are then generated from this transformed representation using pre-defined matrices to account for perceptional factors (we use the matrices from JPEG [1992]).

Before proceeding to the next step of the pipeline, our modified approach goes into the Control component that is further divided into three subcomponents. First, Sampling Control decides where to sample the image with the goal to minimize the total error of a frame with a certain number of samples (Sampling Rate). Second, Compression Control adapts the quality parameter (Compression Rate) such that the specified frame latency is met. Third, Interrupt Control decides whether to continue the refinement of the frame (A) or whether to stop sampling and transfer the frame to the client (B). Sampling Control and Interrupt Control employ regression analysis to estimate current and eventual future errors (Sec. 4), while Compression Control estimates network bandwidth and image size after compression to stay within user-defined latency restrictions (Sec. 3).

Next, these frequencies F are quantized to F_α by rounding from floating point to the nearest integer (Quantization), considering the Compression Rate α: $F_\alpha = \left\lfloor \frac{F}{\alpha} \right\rceil$. In our modified approach, adjusted frequencies from the previously transferred frame are subtracted from the result (Residual Subtraction). The quantized frequencies are then reordered to improve locality and encoded by an entropy encoder (Reordering & Entropy Encoding). The result is then transferred to the client, where the visualization image is finally decoded and displayed. All computation steps of our pipeline are executed on the GPU, with the exception of the CPU-based entropy encoding.

Algorithm 1 Control procedure from Fig. 2. $\mathbf{est}_k$ denotes that function is based on an estimation, with k giving the result type.

```
1:  procedure CONTROL
2:      ▶ Sampling Control
3:      $r = \mathbf{est}_r(\Delta t)$          ▷ estimate number of rays for this iteration
4:      $S_t = \mathbf{sampleGen}_\sigma(r)$          ▷ generate ray sampling tasks
5:      ▶ Compression Control
6:      $[\varsigma, \varsigma^{\rightarrow}] \leftarrow \mathbf{est}_\varsigma(F, [t, t-(\Delta t + o^{\rightarrow})])$          ▷ estimate data sizes
7:      $[\alpha, \alpha^{\rightarrow}] \leftarrow \mathbf{est}_\alpha([\varsigma, \varsigma^{\rightarrow}])$     ▷ quantization factor to yield $[\varsigma, \varsigma^{\rightarrow}]$
8:      ▶ Interrupt Control
9:      $[\sigma_r, \sigma_r^{\rightarrow}] \leftarrow \mathbf{est}_\sigma([\alpha, \alpha^{\rightarrow}], [\emptyset, S_t])$          ▷ $\sigma$ estimation
10:     if $\sigma_r < \sigma_r^{\rightarrow}$ then     ▷ estimated next smaller than current error
11:         continue with Branch A (Fig. 2)
12:     else
13:         continue with Branch B (Fig. 2)
```

3 Control

Control is the core component of our remote visualization approach. It is divided into three phases with different objectives: Sampling Control, Compression Control, and Interrupt Control (Alg. 1). First, in Sampling Control, the upcoming sampling tasks are determined, i.e., a list of sampling positions in image space and the sampling density along the respective rays (Alg. 1, from Line 2). Before these are generated, the number of rays (the so-called ray chunk size) is estimated that yields a given render time of Δt (Line 3). This target render time Δt is set to be a good tradeoff between high granularity (small Δt) which enables higher accuracy for control, and low granularity (large Δt_i) which decreases induced overhead and increases GPU utilization. Then, as many sampling tasks are generated to match this number, and located such that the respective rays improve the image quality as much as possible (Line 4). The number and location of samples required depends on the resolution level l of the respective block (Fig. 3 and right quarter of the renderings in Fig. 1). Blocks are selected for refinement to the next level according to their sorting in terms of anticipated error σ_r (higher errors first).

Compression Control then estimates the quantization factor α (the Compression Rate) to match the target frame time $\bar{t}$ when interrupting the refinement of the frame in this iteration (i.e., take exit B in Fig. 2). Furthermore, to support Interrupt Control, the quantization factor $\alpha^{\rightarrow}$ is estimated that would allow to match $\bar{t}$ after the next iteration. We first determine the respective data sizes ς and $\varsigma^{\rightarrow}$ that would achieve the required time for transfer t_{transfer} for this and the next iteration, respectively, based on bandwidth and latency estimates (Line 6). With this, we then estimate the quantization factors α and $\alpha^{\rightarrow}$ that after compression yield ς and $\varsigma^{\rightarrow}$, respectively (Line 7). In our implementation, we do this by compressing a frame with two different values for α in every third refinement iteration (according to our experiments a good trade-off between overhead computation time and accuracy), and inter-/extrapolate on this basis to estimate the outcome for other values of α.

In Interrupt Control, we use the quantization factors $\alpha, \alpha^{\rightarrow}$ and the information about the upcoming refinement S_t to decide whether to remain in iterative refinement (Branch A, i.e., continue rendering and improving the frame this way), or to exit and send the frame to the client (Branch B). For this, we estimate the errors σ_r and $\sigma_r^{\rightarrow}$ that would result if leaving the refinement process in this or in the next iteration, respectively (Line 9). Then, if the anticipated deviation in the next frame $\sigma_r^{\rightarrow}$ is smaller than that of the current frame σ_r (Line 10), we continue refinement with Volume Raycasting (Branch A, Line 11). Otherwise, refinement of the frame is stopped (Branch B, Line 13), and we continue with compressing it, sending it to the client, and start working on the latest present frame request.

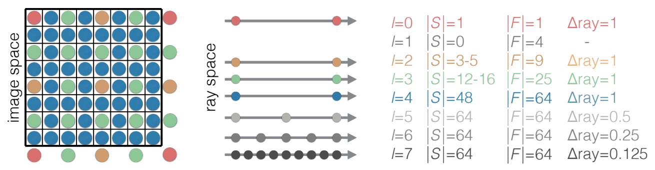
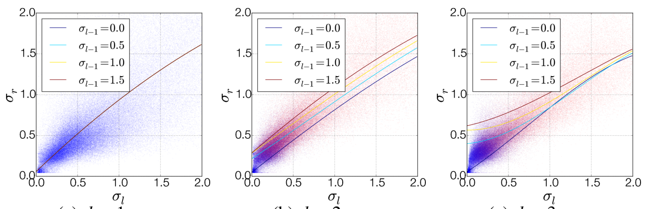


image space
ray space
l=0 |S|=1 |F|=1 Δray=1
l=1 |S|=0 |F|=4 -
l=2 |S|=3-5 |F|=9 Δray=1
l=3 |S|=12-16 |F|=25 Δray=1
l=4 |S|=48 |F|=64 Δray=1
l=5 |S|=64 |F|=64 Δray=0.5
l=6 |S|=64 |F|=64 Δray=0.25
l=7 |S|=64 |F|=64 Δray=0.125

Figure 3: *Sampling patterns for resolution levels l in image and ray space. Samples from adjacent blocks are used for $l = 1, 2$, and 3 to reduce the required additional number of samples $|S|$. From $l \geq 5$, all pixels are sampled with increasingly fine sampling along rays (Δray). $|F|$ gives the number of generated frequency components.*



(a) $l = 1, \alpha_8$ (b) $l = 2, \alpha_8$ (c) $l = 3, \alpha_8$

Figure 4: *Each point in the scatterplots represents the data of one block. Its position depicts the relation between σ_l and σ_r, and the color of a point indicates the value of σ_{l-1} (blue means low value, red stands for large value). The lines show our 3D surface fit (through σ_l, σ_{l-1}, and σ_r or $\sigma_r^{\rightarrow}$) for specific values of σ_{l-1}.*

This approach follows a classic hill climbing scheme, and the characteristic development of the respective values is shown in Fig. 1 for one frame. It can be seen that the frame is interrupted after the curve of σ_r intersects the one of $\sigma_r^{\rightarrow}$ (Iteration 17 in this case).

4 Estimation of Block Deviation

We approximate the quality of the (quantized and unquantized) frequency representation of each block with the standard deviation $\hat{\sigma}_r{}^l$ of the frequencies of the current level l to their reference, i.e., to the unquantized frequencies of this block at the highest resolution level $l_{\max}$ ($l = 7$ in our implementation). To generate an estimate σ_r of $\hat{\sigma}_r{}^l$ that can be used on-the-fly, we collect data of previously rendered frames of representative datasets and views that were fully refined to the highest level $l_{\max}$. From this, we generate a function to predict $\hat{\sigma}_r$ from the determined values for σ_l and σ_{l-1} using regression analysis (σ_l depicts the standard deviation between the frequency representation of the current level l and the previous level $l-1$, while σ_{l-1} stands for the one between $l-1$ and $l-2$). In our experiments, we found that considering both σ_l and σ_{l-1} allows for more accurate estimates (as indicated by the colored point data in the scatterplots in Fig. 4). For fitting, we use a degree-three polynomial with two indeterminates (σ_l and σ_{l-1}) and perform linear ridge regression (also known as Tikhonov regularization). In detail, we utilize a pseudo-Vandermonde matrix, that is composed of $n+1$ measurements of σ_l and σ_{l-1}. For Sampling Control, we fit the unquantized version of both the current and the upcoming block resolution level (σ_r and $\sigma_r^{\rightarrow}$). For Interrupt Control, we generate fits for a total of 16 different quantization levels $\sigma_{r\alpha}$: $\alpha_0, \ldots, \alpha_{15}$, with $\alpha_0 = 0.02$ and $\alpha_{15} = 10$, and the other values distributed evenly in between (i.e., between 5 and 99 in terms of JPEG's quality definition [JPEG 1992]). A selection of fitting results is depicted in Fig. 4. For generating an estimate with an arbitrary α, with $\alpha_k \leq \alpha < \alpha_{k+1}$, we perform linear interpolation between α_k and α_{k+1} to reach the final result. We do this, as according to our observations, the transitions between the levels is smooth, and it allows us to reduce the complexity involved in the fitting and the evaluation process.

5 Results

We conducted a comprehensive evaluation covering different system setups, methodical variants and five different datasets: Chameleon ($1024^2 \times 1080$), Mouse ($1024^2 \times 975$), Supernova (432^3), Lambda (529^3), and Rayleigh-Taylor ($128^2 \times 256$) (Fig. 5). For each dataset, we created a short camera path that represents standard user interaction sessions. Two full-reference metrics are used for image quality estimation: multi-scale structural similarity (MSSSIM, value range $[0, 1]$, larger is better) [Wang et al. 2003] and peak signal-to-noise ratio (PSNR, value range $[0, \infty)$, larger is better). They are particularly suited for comparison of frames belonging to the same setup, but absolute values have to be used with caution across different configurations (e.g., [Huynh-Thu and Ghanbari 2008]). Since a reference image is required for both metrics, we run our visualization procedure twice for each configuration (one online run in real time, and one offline run to generate the respective full-quality images). Throughout our evaluation, we used an image resolution of 1024×600. On the GPU of our server machine (NVIDIA GeForce GTX Titan, Intel Core i7 X980), generating sampling tasks took around 0.18 ms, the transformation of samples to the frequency domain 0.11 ms, and the quantization, residual subtraction, and reordering of frequencies to prepare them for compression took around 0.4 ms. Downloading the result to the CPU took 0.7 ms. The compression of a frame using LZ4 on the CPU required approximately 5 ms. On the client (NVIDIA GTX680, Intel Core i7-3820), LZ4 decompression took 2.5 ms, and the reconstruction of a frame for display took around 1 ms. Note that values may vary depending on various factors (e.g., block resolution levels). We target a total frame latency of 0.15 ms throughout our evaluation.

Sampling Control. We evaluated different criteria for selecting blocks for refinement: $s(\sigma_r)$ (our prediction), $s(\sigma_l)$ (error to the previous level), $s(\sigma_{l-1} - \sigma_l)$ (gradient of error development, cf. [Bolin and Meyer 1995]), and $s(\hat{l} - l)$ (blocks of the same level have the same value). Here, quality loss due to lossy quantization is not considered. Tab. 5a shows that $s(\sigma_r)$ produces the best results, although followed closely by $s(\sigma_l)$, with $s(\hat{l} - l)$ performing the worst. Note that $s(\sigma_l)$ is relatively close to $s(\sigma_r)$ for low resolution levels l, with the difference becoming larger the higher the sampling density gets (cf. Fig. 4). Generally, the more time is available for rendering, the better $s(\sigma_r)$ performs in comparison to the other modalities, as the variance in the predictions decreases. Further, for datasets with high visual complexity (e.g., the Chameleon), good decisions on where to refine have a strong impact, while particularly for the smoother Rayleigh-Taylor, but also the Supernova dataset, the difference is much smaller. In Fig. 6a, $s(\hat{l} - l)$ leads to blurry results over wide regions of the dataset, these regions are much more detailed in (b) as well as closer to the reference in (c).

Compression Control. In the following, the bandwidth is restricted to different values to simulate different remote visualization setups: 2, 16, and 64 MB/s. Our approach flexibly adapts compression quality to systems with different performance characteristics, and in terms of quality values PSNR and MSSSIM performs better than the tuned fixed-quality settings (Tab. 5b). The fixed settings (10, 50, 90) are relatively close in value to what our adaptive approach achieves for low, medium, and high bandwidth settings, respectively (for 2 MB/s, high fixed-quality settings often significantly exceed the target latency time). Nevertheless, our quality results are significantly better even if the fixed quality suits, as we adapt each frame individually based on its content. Fig. 6 further shows that our adaptive compression (e) introduces the flexibility to find a good trade-off between sampling and compression, with more visible details than

(a) Sampling Control

Setting	Time (s)	MSSSIM	PSNR	σ_r	Render	Transfer
$s(\sigma_r)$	0.04	0.907	28.884	0.448	0.044	0.0
$s(\sigma_l)$	0.04	0.907	28.868	0.448	0.044	0.0
$s(\sigma_{l-1}-\sigma_l)$	0.04	0.905	28.763	0.471	0.045	0.0
$s(\hat{l}-l)$	0.04	0.897	28.346	10.801	0.044	0.0
$s(\sigma_r)$	0.08	0.919	29.467	0.342	0.083	0.0
$s(\sigma_l)$	0.08	0.917	29.297	0.35	0.083	0.0
$s(\sigma_{l-1}-\sigma_l)$	0.08	0.915	29.051	0.4	0.084	0.0
$s(\hat{l}-l)$	0.08	0.918	28.88	16.325	0.084	0.0
$s(\sigma_r)$	0.12	0.941	31.74	0.248	0.124	0.0
$s(\sigma_l)$	0.12	0.935	31.147	0.27	0.124	0.0
$s(\sigma_{l-1}-\sigma_l)$	0.12	0.929	30.19	0.359	0.125	0.0
$s(\hat{l}-l)$	0.12	0.932	29.759	15.521	0.126	0.0

(b) Compression Control

Setting	Bw.	MSSSIM	PSNR	σ_r	Render	Transfer
10	2	0.897	29.03	0.436	0.099	0.053
20	2	0.898	29.148	0.422	0.076	0.076
90	2	0.878	28.229	0.532	0.031	0.144
a. (25.7)	2	0.914	30.49	0.338	0.082	0.084
10	16	0.916	30.021	0.45	0.143	0.008
20	16	0.928	30.86	0.333	0.137	0.011
90	16	0.919	30.196	0.319	0.117	0.036
a. (58.4)	16	0.942	32.097	0.228	0.132	0.023
10	64	0.917	30.113	0.455	0.147	0.003
20	64	0.931	31.087	0.334	0.145	0.004
90	64	0.926	30.744	0.289	0.137	0.014
a. (81.6)	64	0.947	32.621	0.199	0.142	0.013

(c) Interrupt Control

Setting	Bw.	MSSSIM	PSNR	σ_r	Render	Transfer
$\sigma_r < \sigma_r^{\rightarrow}$	2	0.919	30.495	0.359	0.083	0.092
$0.5\sigma_r < \sigma_r^{\rightarrow}$	2	0.929	31.038	0.445	0.118	0.08
$2\sigma_r < \sigma_r^{\rightarrow}$	2	0.857	27.581	0.597	0.024	0.149
$t < 0.5\bar{t}$	2	0.921	30.383	0.347	0.082	0.079
$\sigma_r < \sigma_r^{\rightarrow}$	16	0.943	32.152	0.228	0.131	0.023
$0.5\sigma_r < \sigma_r^{\rightarrow}$	16	0.944	32.364	0.348	0.146	0.014
$2\sigma_r < \sigma_r^{\rightarrow}$	16	0.921	30.31	0.304	0.085	0.054
$t < 0.5\bar{t}$	16	0.925	30.487	0.293	0.087	0.052
$\sigma_r < \sigma_r^{\rightarrow}$	64	0.947	32.64	0.2	0.142	0.013
$0.5\sigma_r < \sigma_r^{\rightarrow}$	64	0.944	32.444	0.343	0.153	0.006
$2\sigma_r < \sigma_r^{\rightarrow}$	64	0.939	32.108	0.224	0.126	0.021
$t < 0.5\bar{t}$	64	0.925	30.505	0.292	0.087	0.02

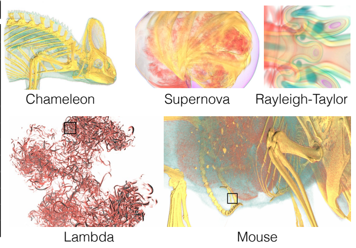


Figure 5: *Averaged results over different datasets. Closeups for selected regions (black rectangle) are shown for lambda and mouse in Fig. 6.*

with fixed low (d) or high (f) compression quality.

Interrupt Control. Our standard $\sigma_r < \sigma_r^{\rightarrow}$ is compared against $0.5\sigma_r < \sigma_r^{\rightarrow}$, and $2\sigma_r < \sigma_r^{\rightarrow}$, as well as against a fixed predefined time share for rendering and transfer $t < 0.5\bar{t}$ (spending half of the specified frame latency for rendering, and the other half for transfer). Regarding $\sigma_r < \sigma_r^{\rightarrow}$, $0.5\sigma_r < \sigma_r^{\rightarrow}$, and $2\sigma_r < \sigma_r^{\rightarrow}$, the impact of using a scheme with a modified criterion can already be estimated from Fig. 1 (right): $0.5\sigma_r < \sigma_r^{\rightarrow}$ typically leaves refinement too early according to our predictions, while $0.5\sigma_r < \sigma_r^{\rightarrow}$ interrupts too late. The fixed variant $t < 0.5\bar{t}$ is not able to adapt to the visualization process. Overall, $\sigma_r < \sigma_r^{\rightarrow}$ delivers the best results, considering both quality as well as compliance with the total computation time limit (Tab. 5c). For the lowest bandwidth, variants forcing a relatively high ratio of render time significantly exceed the target latency in this case ($0.5\sigma_r < \sigma_r^{\rightarrow}$), as a small enough transfer size cannot be achieved anymore. For the low bandwidth setting, $\sigma_r < \sigma_r^{\rightarrow}$, $0.5\sigma_r < \sigma_r^{\rightarrow}$, and $2\sigma_r < \sigma_r^{\rightarrow}$ also sometimes exceed the total limit of 0.15 ms, which we attribute to inaccuracies in our bandwidth estimation, as the variation can be large particularly for low iterations counts.

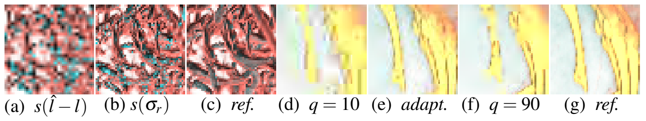
6 Conclusion and Future Work

We presented a novel approach for handling sampling and compression in remote visualization in an integrative fashion, and showed its utility for a range of datasets and bandwidth settings. It adaptively prioritizes regions for refinement with a novel progressive rendering scheme (Sampling Control), dynamically chooses the quantization quality for compression to meet latency restrictions (Compression Control), and determines the best time share for visualization and transfer for each frame on-the-fly (Interrupt Control). We employ regression analysis-based error prediction considering both undersampling and lossy compression for driving the decision-making.

For future work, Interrupt Control could be combined with previous work [Frey et al. 2014] in which we adjusted the time a frame may take (in contrast to the fixed latency setting in this paper) to balance the errors in local rendering due to undersampling and temporal changes. Conducting a user study on top of our quality metric-based evaluation could further lead to additional insights. Also, currently, our prototype implementation uses a JPEG-based compression scheme for the sake of simplicity, which could be changed into a video codec-based streaming for direct performance comparison to state-of-the art remote visualization applications. We further anticipate that we might be able to use hardware video functionality (as introduced by Intel, AMD, and NVIDIA [2014]) in case their flexibility further increases (supplying the intermediate frequency representation, in particular). Finally, our error estimation per block could be used for adaptive compression in image space.

Acknowledgements

The authors thank the German Research Foundation (DFG) for financial support within project A02 of SFB/Transregio 161.



(a) $s(\hat{l}-l)$ (b) $s(\sigma_r)$ (c) *ref.* (d) $q = 10$ (e) *adapt.* (f) $q = 90$ (g) *ref.*

Figure 6: *Closeups of Lambda (a)–(c) and Mouse (d)–(g) (Fig. 5).*

References

BOLIN, M. R., AND MEYER, G. W. 1995. A frequency based ray tracer. In *22nd annual conference on Computer graphics and interactive techniques*, ACM, SIGGRAPH, 409–418.

DIEPSTRATEN, J., GORKE, M., AND ERTL, T. 2004. Remote line rendering for mobile devices. In *Proc. Computer Graphics International*, 454–461.

FREY, S., SADLO, F., MA, K.-L., AND ERTL, T. 2014. Interactive progressive visualization with space-time error control. *IEEE Transactions on Visualization and Computer Graphics 20*, 12, 2397–2406.

HERZOG, R., KINUWAKI, S., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2008. Render2MPEG: a perception-based framework towards integrating rendering and video compression. *Computer Graphics Forum 27(2)*, 183–192.

HUYNH-THU, Q., AND GHANBARI, M. 2008. Scope of validity of PSNR in image/video quality assessment. *Electronics Letters 44*, 13, 800–801.

JPEG, 1992. ISO/IEC IS 10918-1, ITU-T Recommendation T.81.

KOLLER, D., TURITZIN, M., LEVOY, M., TARINI, M., CROCCIA, G., CIGNONI, P., AND SCOPIGNO, R. 2004. Protected interactive 3D graphics via remote rendering. ACM, SIGGRAPH, 695–703.

LEVOY, M. 1990. Volume rendering by adaptive refinement. *The Visual Computer 6*, 1, 2–7.

MA, K.-L., AND CAMP, D. M. 2000. High performance visualization of time-varying volume data over a wide-area network. In *ACM/IEEE conference on Supercomputing*.

MORELAND, K., LEPAGE, D., KOLLER, D., AND HUMPHREYS, G. 2008. Remote rendering for ultrascale data. *Journal of Physics: Conference Series 125*, 1, 012096.

NVIDIA, 2014. Nvidia video encoder. developer.nvidia.com.

PAJAK, D., HERZOG, R., EISEMANN, E., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2011. Scalable remote rendering with depth and motion-flow augmented streaming. *Computer Graphics Forum 30*, 2, 415–424.

WANG, Z., SIMONCELLI, E., AND BOVIK, A. 2003. Multiscale structural similarity for image quality assessment. In *Proc. Signals, Systems and Computers*, vol. 2, 1398–1402.