Spatiotemporal LOD-Blending for Artifact Reduction in Multi-Resolution Volume Rendering

Sebastian Thiele*　Carl-Feofan Matthes†　Bernd Froehlich‡

Virtual Reality and Visualization Research Group
Bauhaus-Universität Weimar

Abstract

High-quality raycasting of multi-resolution volumetric datasets benefits from a well-informed working set selection that accounts for occlusions as well as output sensitivity. In this work, we suggest a feedback mechanism that provides a fine-grained level-of-detail selection for restricted working sets. To mitigate multi-resolution artifacts, our rendering solution combines spatial and temporal level-of-detail blending to provide smooth transitions between adjacent bricks of differing levels of detail and during working set adjustments. We also show how the sampling along rays needs to be adapted to produce a consistent result. Our implementation demonstrates that our spatiotemporal blending in combination with consistent sampling significantly reduces visual artifacts.

1 Introduction

The visualization of massive models exceeding the size of internal memories requires out-of-core multi-resolution rendering algorithms, designed to effectively reduce the amount of data loaded and considered during rendering [7]. However, out-of-core rendering systems frequently compromise the visual quality of the rendering result [3].

We present a set of novel methods developed for high-quality ray-guided real-time out-of-core visualization of large volumetric data sets. Disturbing popping artifacts during working set changes and salient discontinuities at brick boundaries are mitigated by a spatiotemporal interpolation scheme, which combines trilinear interpolation with spatial and temporal level-of-detail blending. Since our blending approach requires a restricted working set, we add two conditions for split and collapse operations during level-of-detail selection. Frequently, ray-guided feedback mechanisms are employed to inform the working set selection process and optimize resource utilization [4, 6]. During raycasting, we accumulate feedback information for visible bricks of the working set and derive priorities to inform our level-of-detail selection scheme.

Our main contributions are:

- an efficient blending approach that combines spatial and temporal level-of-detail blending to overcome the most salient multi-resolution artifacts,
- a consistent adaptive sampling technique that avoids oversampling as well as abrupt changes of the sampling stepsize, and
- a low-overhead level-of-detail feedback mechanism and an efficient priority distribution scheme to guide the working set selection process.

*e-mail: sebastian.thiele@uni-weimar.de
†e-mail: carl-feofan.matthes@uni-weimar.de
‡e-mail: bernd.froehlich@uni-weimar.de

We integrated all proposed techniques into a multi-resolution volume raycasting framework and demonstrate the improved rendering quality of our approach.

2 Related Work

Multi-resolution volume rendering is a well-researched field and in this paper we limit our discussion to the work most closely related to our contributions. Beyer et al. [1] provide an overview of state-of-the-art volume rendering.

Multi-resolution approaches for the visualization of large volumetric datasets display the volume at locally varying levels of detail (LOD) and employ an octree representation of the dataset to store the working set in a texture atlas [7, 4, 5]. The *working set* is the result of a level-of-detail selection process and constitutes the set of bricks available during rendering. A brick consists of a fixed number of voxels (e.g. 64^3) and serves as the paging unit. The leaf-level of the working set is also referred to as the *cut*. Commonly, view-dependent LOD cut updates are incrementally performed using a greedy-style split-and-collapse algorithm similar to [2], maximizing the data reuse by exploiting frame-to-frame coherence. In our work, the working set is *restricted* by adjacency relationships between bricks and we demonstrate how two conceptually simple conditions suffice to maintain a restricted working set on a frame-to-frame basis.

Ideally, a feedback mechanism is used to gather information about required data bricks directly during rendering. This is central to the output sensitivity of the system and greatly facilitates a restriction of the working set to the bricks actually visible. Crassin et al. [4] use render targets to keep track of brick utilization and store a single bit to indicate whether a given brick should be split. Hadwiger et al. [8] store feedback information in form of cache misses and usage statistics atomically in a set of feedback-tiles shared with other rays through screen-space partitioning directly during rendering. Fogal et al. [6] implement a lock-free set datastructure to store cache misses. However, the feedback generated in previous systems is limited to binary information about whether bricks should be split or collapsed and does not involve per-brick priorities gathered during sampling. To provide a fine-grained prioritization during working set selection, we store level-of-detail differences in our feedback table and provide a priority distribution algorithm connecting the feedback mechanism with a split-and-collapse working set selection strategy.

There are a number of artifacts commonly arising in multi-resolution volume raycasting systems. Carmona et al. [3] observe that visible *boundary artifacts* between bricks of differing levels of detail can be reduced through *spatial level-of-detail blending* with the parent brick towards boundaries of coarser neighboring bricks. Ljung et al. [9] interpolate boundary voxels between adjacent bricks, leaving the boundary discontinuity between bricks quite noticeable. Furthermore, frame-to-frame working set adjustments in out-of-core scenarios often cause *popping artifacts* during split and collapse operations. We introduce a novel *spatiotemporal level-of-detail blending* algorithm by generalizing the work of Carmona et al. [3], who employ a binary weight at each ver-

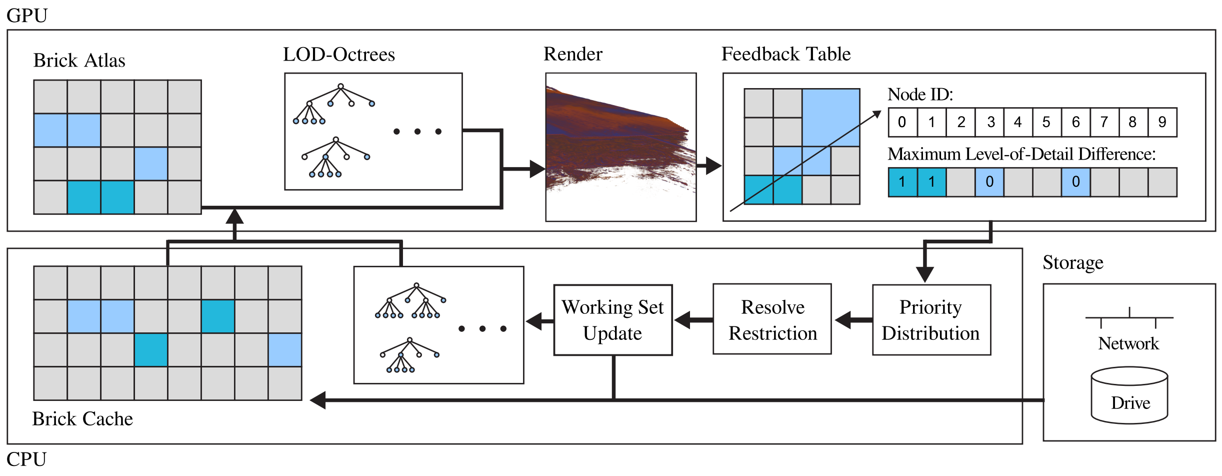


Figure 1: This Figure depicts a high-level overview of our system. Each time a ray begins sampling a brick during rendering, the difference between the ideal level of detail at the sampling position and the level of detail actually available is stored in a feedback table. Next, we distribute derived priorities for the entire working set from the subset of bricks for which the priority was obtained through the feedback mechanism. During the resolve restriction step, we guarantee a restricted working set every frame and implement the adjacency constraint through forced split-operations. The actual working set update controls asynchronous data-delivery from external storage.

tex. In this work, we introduce continuous spatiotemporal weights which are updated on a frame-to-frame basis to provide for both animated transitions during working-set changes and gradual transitions across adjacent bricks of differing LODs.

Commonly, the sampling density is adapted to match the brick resolution, e.g. [7]. However, in order to obtain a consistent rendering result in our approach, we incorporate additional considerations into the computation of the sampling density such as spatial and temporal blending as well as the ideal level-of-detail.

3 Direct Level-of-Detail Feedback

Performance and resource utilization benefit from a well-informed LOD selection that accounts for occlusions and output sensitivity. Volume rendering techniques that incorporate per-brick information gathered during sampling to drive the LOD selection process are referred to as *ray-guided* [6].

Our out-of-core multi-resolution volume rendering framework [11] is able to interactively handle massive volumetric datasets in the terabyte range. Figure 1 illustrates our basic system architecture. Similar to [7], our out-of-core data management system is based on a two-level cache hierarchy and consists of three main components: The brick cache, the brick atlas and the level-of-detail feedback mechanism.

3.1 Feedback Table

During rendering, we populate a feedback table of brick usage statistics that is shared across all rays and transferred from graphics memory to main memory in order to guide our LOD selection process. Before rendering, we reserve a feedback entry for each brick present in the working set. In order to converge on the best possible working set for rendering, we store the maximum level-of-detail difference required by all rays that sample a brick in the feedback table. Similar to [8], we index the location of feedback information based on the corresponding brick id which allows for direct feedback accumulation without an additional histogram compaction step.

We initialize all entries in our feedback table P with $-\infty$. Whenever a ray begins sampling a brick, we atomically store the maximum of all differences of the depth of brick N to the optimal LOD at the given sampling position in the feedback table:

$$P_N = \max\left\{LOD_{actual}(N,R) - LOD_{ideal}(N,R) \mid R \in \mathscr{R}\right\} \quad (1)$$

where $\mathscr{R}$ is the set of rays that sample brick N. We also keep a record of the number of rays that have sampled each brick to augment our working set selection prioritization.

We opt to serialize the layout of our level-of-detail octree in graphics memory instead of using an index texture, because at raycasting time we stop traversing the working set as soon as the ideal level-of-detail is reached. This guarantees that we never oversample bricks and our restricted working set does not cause overdraw. In order to ensure availability of data from ancestors of any brick selected for rendering, all ancestors of bricks present in the cut are kept in the atlas texture in graphics memory and are part of the working set. This approach allows for sampling of inner bricks of the working set as well as immediate collapses during the cut update.

3.2 Feedback-guided LOD Selection

Given the most recent feedback table downloaded to main memory, we initiate our LOD selection scheme by performing a sweep over the working set. Over the course of this traversal, we compute derived priorities for all nodes in the working set from the subset of known priorities contained in the feedback table.

First, we obtain priority P_N corresponding to node N from the feedback table. We define P_N to be the maximum of $LOD_{actual}(N,R) - LOD_{ideal}(N,R)$ for all rays R that sample brick N (Section 3.1). Note that P_N is positive for nodes that are too coarse and zero if the LOD of N is ideal. If there is no priority for node N available in the feedback table, then $P_N = -\infty$. Next, we traverse the entire working set top-down breadth-first and propagate priorities from the feedback table (cmp. algorithm in Algorithm 1). During the bottom-up traversal, we store the corresponding value of P_N of each node N.

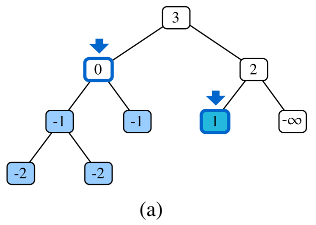
After the priority distribution, the priority P_A for any given node A is larger than P_N for ancestors of N, and smaller than P_N for descendants of N (Figure 2). Over the course of the LOD selection process, nodes in the leaf level of the working set for which $P_N < 0$ are queued for collapse because their LOD is considered too fine by all rays. In contrast, any nodes at the leaf level of the working set

Algorithm 1 Priority distribution

```
1:  Input
2:      Working Set $W$
3:      Feedback Table $F$
4:
5:  Queue $Q \leftarrow$ push root of $W$
6:  Queue $T$
7:  // top to bottom propagation
8:  while ($Q$ not empty)
9:      $N \leftarrow$ pop($Q$)
10:     for each child $C$ of $N$
11:         $P_C \leftarrow \max(F(C), P_N - 1)$
12:         $Q \leftarrow$ push $C$
13:     if (children of $N$ have no children in $W$)
14:         $T \leftarrow$ push $N$
15: // bottom to top traversal
16: while ($T$ not empty)
17:     $N \leftarrow$ pop($T$)
18:     $P_N \leftarrow -\infty$
19:     for each child $C$ of $N$
20:         $P_N \leftarrow \max(P_N, P_C + 1)$
21:     if ($N$ not root of $W$)
22:         $T \leftarrow$ push parent($N$)
```

for which $P_N > 0$ are candidates for splitting as their LOD is considered too coarse by some rays. Nodes that are occluded or outside of the viewing-frustum receive a negative priority and are therefore subject to collapse.

During the split-and-collapse working set update, split operations are performed in accordance with the per-brick priorities established above. If two bricks have equal priority, then we consider the number of rays that sampled the bricks in question to augment our prioritization. Thus, we achieve a fine-grained prioritization during the LOD selection, which is used to adhere to global memory constraints and memory transfer budgets. Our priorities are suitable for the prioritization of the out-of-core data delivery queue in external memory scenarios as well.



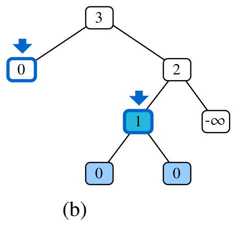


Figure 2: Illustration of the propagation of priorities across the working set in our approach, depicted as a binary tree. Feedback information is obtained for two nodes marked with arrows. The resulting propagated priorities after the working set traversal are shown in (a). Over the course of the working set update, nodes with negative priority are collapsed and nodes with a positive priority are split. The updated working set is shown in (b).

4 Spatiotemporal Level-of-Detail Blending

Abrupt changes in resolutions between adjacent bricks cause salient discontinuities that domain experts considered distracting. We mitigate these cross-block artifacts using an approach that extends the one described in [3] where bricks are gradually interpolated with data from their parent in the octree as rays are sampled towards the boundary of coarser neighbors. Disturbing popping artifacts during working set changes remain visible in [3] and we generalize their binary spatial weights to continuous spatiotemporal weights to provide for visually pleasing transitions over time during working set adjustments. In this approach, the working set selected for rendering is restricted such that the difference with respect to the LOD between adjacent bricks does not exceed one level.

4.1 Restricted Working Set Selection

To maintain a restricted working set throughout LOD selection, we include additional conditions for split and collapse operations. The adjacency constraint dictates that node N can be split only if $(\text{depth}(N) - \text{depth}(B_i)) < 1$ holds for all its neighbors B_i. As long as there are neighbors that are too coarse, the split cannot be performed. In order to implement this assertion, we keep a record of adjacent bricks for each node to perform lookups of adjacent neighbors in the current working set. Note that we store only the six direct neighbors per node, as the remainder of at most 26 neighbors are inferred using the references stored in the adjacent nodes.

All neighbors that are too coarse to allow for a split are collected in an auxiliary stack. Nodes residing on this stack need to be split before the restricted split of node N is valid. At this point, the adjacency constraint has to be re-evaluated for any node in the auxiliary stack, eventually adding additional nodes to the auxiliary stack that prevent splits of N's neighbors. This process repeats until no more nodes violating the adjacency constraint are encountered. Nodes with no dependencies are split first, and node N is split last. However, the number of nodes that are split on a per-frame basis must not exceed a pre-defined memory transfer budget and occasionally, it may not be possible to split node N in the current iteration of the working set selection.

Priority-based collapse operations of a node N are only executed if all of its neighbors B_i fulfill $(\text{depth}(N) - \text{depth}(B_i)) \geq -1$. In contrast to split operations, if there are neighbors preventing the restricted collapse of node N, we do not force collapses of these neighbors. This approach introduces an asymmetry or hysteresis between split and collapse operations in the working set selection which stabilizes the working set and significantly reduces flickering between states.

In our implementation, we restrict the entire working set. In theory, bricks that are occluded or outside the viewing-frustum do not have to be subjected to restriction. However, as soon as previously occluded bricks become visible or move into the viewing frustum, not having restricted them before causes noticeable artifacts.

4.2 Temporal Level-of-Detail Blending

Our renderer animates transitions between LODs during split and collapse operations. We associate a *primary temporal weight* t_N with each node in the hierarchy, indicating the interpolation between the parent and the node itself. During split operations, we gradually increase t_N over time from 0.0 to 1.0 at which point the node is fully visible. Similarly, we decrease t_N during collapse operations. All nodes for which $t_N > 0.0$ are part of the working set. Furthermore, we guarantee that the outcome of any temporal transitions pending do not invalidate the restriction of our working set.

4.3 Spatial Level-of-Detail Blending

Carmona et al. [3] use one bit at each vertex of a brick to indicate whether any adjacent brick is coarser than the brick in question. During rendering, they consider eight vertex bits for any given sampling position to determine the influence of the parent of the brick being sampled. If at least one adjacent brick is coarser than brick N, samples taken from brick N and its parent in the LOD hierarchy are linearly interpolated to obtain the final sample and provide for smooth transitions between bricks of differing levels of detail (Figure 3).

In our work, we generalize the binary weight at each vertex to a continuous *spatiotemporal weight* that is used to adapt gradually

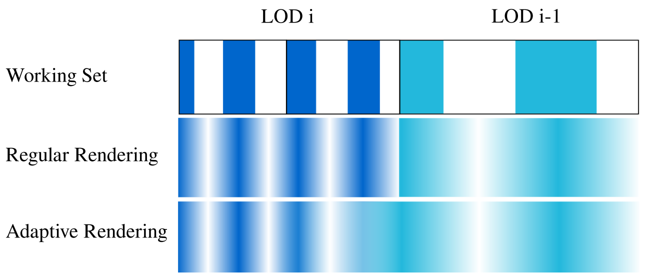


Figure 3: Illustration of spatial level-of-detail blending. The working set (top) contains bricks of differing level of detail. A regular rendering of these bricks comes with salient boundary artifacts (middle). An adaptive rendering strategy provides for higher visual fidelity in theses cases (bottom).

between the brick being sampled and its parent (Figure 4). The spatiotemporal weight indicates the minimum time step of all nodes of equal depth adjacent to the corresponding vertex. As soon as any brick adjacent to the vertex is coarser than the brick in question, we set the corresponding spatiotemporal weight to zero. Otherwise, the spatiotemporal weight s_v of vertex v for brick N is given by $s_v = \min(t_{B_i}),\ i < 8$ where t_{B_i} represent the primary temporal weights of all bricks of equal depth sharing vertex v. Whenever a node in the working set is split or collapsed, the spatiotemporal weights of all affected neighbors need to be updated accordingly. This ensures that temporal transitions of bricks during working set adjustments correctly composite with spatial blending between bricks of differing levels of detail and provides for visually continuous results regardless of the LOD chosen for rendering and ongoing temporal transitions.

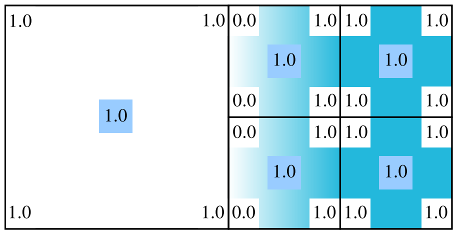


Figure 4: This Figure illustrates spatiotemporal blending for a quadtree along with the configuration of spatiotemporal weights s_v shown in white squares and primary temporal weights t_N in blue squares. When the right brick is split all weights are initialized to zero, indicating that samples are taken from the parent brick only. Over time, we increase the weights, gradually adding to the influence of the child bricks.

5 Adaptive Sampling for Spatiotemporal Blending

In general, sampling the volume at a constant stepsize may cause severe oversampling or undersampling because it is not view-dependent and does not account for local changes in level-of-detail. In our experience, abrupt changes of the stepsize during sampling may cause visible artifacts. Therefore, we believe that an artifact-optimized adaptive sampling strategy must avoid abrupt changes in the sampling stepsize. Consequently, we consider a number of influences to compute an adaptive stepsize at a given sampling position. These influences include the level of detail of the brick being sampled as well as the ideal continuous level of detail at the sampling position. To adapt the sampling density along rays to our spatiotemporal LOD-blending approach, we also incorporate temporal and spatial LOD blend weights in our adaptive stepsize computation. Our particular transfer function (transparency highest in the middle of the transfer function domain) requires the use of the Gauss filter approach for appearance preserving LOD rendering as suggested in [12]. However, this technique cannot be easily combined with pre-integrated volume rendering since it already uses a 2D transfer function lookup. For this reason, we use adaptive sampling in combination with opacity correction.

For perspective projections, the ideal level of detail decreases as the distance between the sampling position and the view point increases. To avoid oversampling or undersampling, the level of detail should be related to the footprint of the pixel at the sampling location, e.g. the diameter of the pyramid created by the pixel and viewpoint at the sampling location. Since the edge length of voxels increases by a factor of two between octree levels, however, voxels with an ideal edge length of v_{ideal} do not usually exist in our octree. We can assume that the smallest voxels have an edge length of one and thus, we define the fractional ideal $LOD_{\text{ideal}} = MaxOctreeDepth - log_2(v_{\text{ideal}})$. For consistency reasons, we set the sampling distance along a ray to v_{ideal} for a voxel to pixel size ratio of 1:1. In case the brick of the ideal level of detail is not present in the working set, we adapt the sampling stepsize to the available brick being sampled (Figure 5(b)), which is usually the highest LOD available in the current working set.

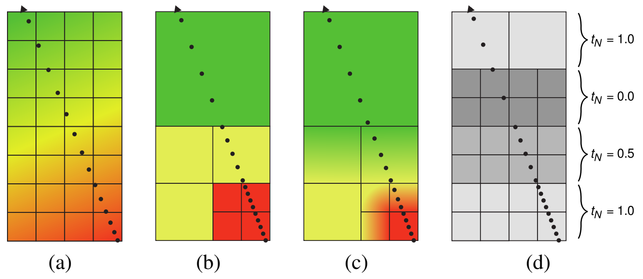


Figure 5: Our adaptive sampling strategy considers four influences: the ideal LOD at the sampling position (a), the actual LOD available in graphics memory (b), the spatial blending between bricks of differing LODs (c) and the primary temporal blendweights for animated transitions during working set adjustments (d). In these illustrations, the color red corresponds to the highest, green to the lowest level of detail and the viewpoint is assumed to reside in the lower right corner.

In addition, we incorporate spatiotemporal LOD-blending as indicated by the corresponding blend weights into our stepsize computation since the stepsize along a ray should be consistent with the fractional LOD required for spatiotemporal blending. Each brick comes with eight spatiotemporal weights as well as one primary weight, which are used to transition between the stepsizes required for the brick being sampled and its parent brick. During a temporal transition, we trilinearly interpolate all eight spatiotemporal weights of the brick being sampled to determine the influence of the parent brick at the current sampling position. In essence, the result of the trilinear interpolation of spatiotemporal weights is used to increase the sampling stepsize smoothly as we sample the ray towards the boundary of a coarser neighbor. Starting from the sampling density required by the brick being sampled, we gradually increase the sampling stepsize up to the sampling density required by the parent brick (Figure 5(c)). The primary temporal weight t_N of any adjacent brick factors into the computation of the spatiotemporal weight of a vertex (Section 4.3). Therefore, any pending temporal transitions correctly blend into the resulting stepsize and require no extra handling.

As a result, we need to consider three cases with respect to the current sampling position and the stepsize computation. First, if there is spatial or temporal blending active for the current brick being sampled, we compute the stepsize from the fractional LOD that corresponds to the interpolated blend weight at the current sampling location. Second, if the available LOD is coarser than the ideal LOD, then we use the sampling stepsize that corresponds to available LOD. Finally, if the ideal LOD is available, which requires that $\lceil LOD_{\text{ideal}} \rceil$ is in the working set, we can use the corresponding stepsize and interpolate appropriately between the corresponding bricks based on the fractional part of LOD_{ideal}. As evident from Figure 6, artifacts along brick boundaries are mitigated using this scheme.

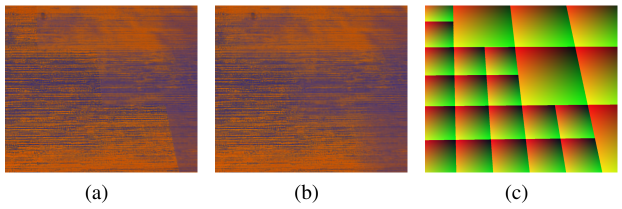


Figure 6: Cross-brick artifacts along the boundary between bricks of differing LODs without LOD blending and adaptive sampling (a). Our spatiotemporal LOD blending and the adaptive sampling scheme provide for smooth, almost imperceptible transitions in these cases by gradually increasing the influence of the parent brick towards the boundary of coarser neighbors (b). The current working set configuration is shown in (c).

6 Results and Discussion

The evaluation was conducted on an *Intel Xeon* CPU with 6 cores at 3.5 GHz, 128 GB main memory along with a *Nvidia GTX Titan X* graphics card with 12 GB video memory. Large datasets in the oil and gas domain are mostly confidential. Fortunately, we received permission to use a seismic dataset from an oil field located in New Zealand for publications. The dataset is 5989x3933x1501 voxels in size and was processed using a bricksize of 64^3 voxels. Given an octree depth of 7, its total size is 87.3 GB and contains 16 bit Gaussian coefficients per voxel as contributed in [12] to overcome inconsistency artifacts during pre-filtering and down-sampling of the dataset.

Our working set generation consists of three steps. Over the course of the priority propagation, we determine a priority for every brick currently residing in the working set. Next, we resolve any dependencies arising from the adjacency constraint to maintain a restricted working set. Finally, the actual working set update is accomplished using an greedy-style split-and-collapse algorithm similar to [2]. As shown in Figure 7, these three computational steps perform in the range between 4 and 8 ms in total for the majority of frames during a simple rotation scenario of the seismic dataset.

We demonstrate how the sampling along rays needs to be adapted to produce a consistent result for our spatiotemporal blending solution as shown in Figure 8. For spatial level-of-detail blending and during temporal transitions, we need to take one additional sample from the parent brick per iteration. However, this data access is highly coherent and in both cases the sampling stepsize is increased in accordance to the influence of the parent brick, resulting in a limited overall sampling overhead in the range of 10 to 20% in our experience (Figure 7). This is also consistent with what was reported by Carmona et al. [3] who performed only spatial blending between LODs.

It is conceivable to prevent cross-block boundary artifacts by trilinearly interpolating voxels within a certain neighborhood of the boundary with multiple ancestors in arbitrary, non-restricted working set configurations. However, data-based working set selection

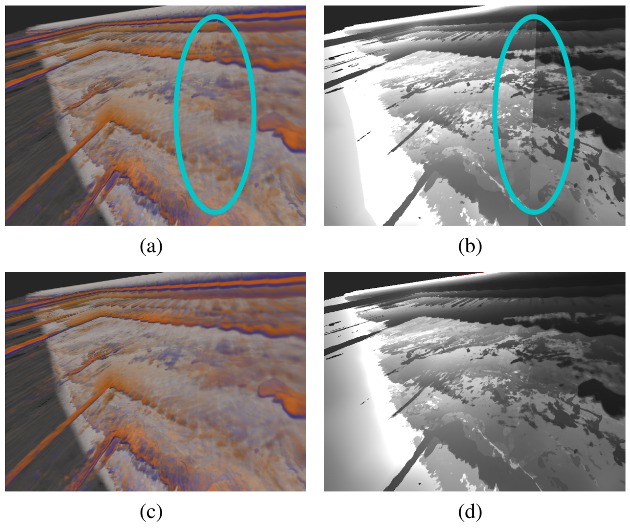


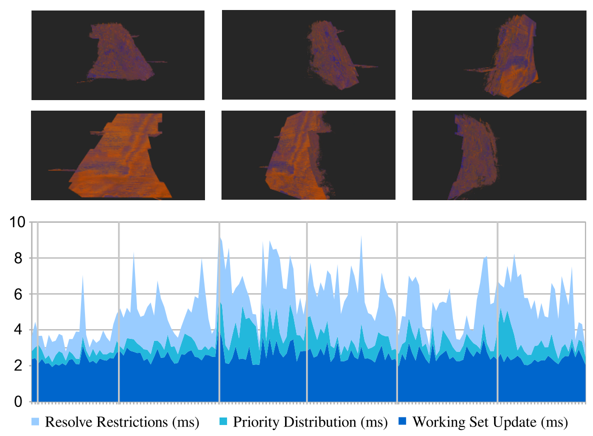
Figure 8: Visualization of per-pixel iteration count during sampling of the seismic dataset. The state-of-the-art sampling approach without spatiotemporal blending shown in (a) and (b) requires a considerable number of sampling operations and produces salient artifacts (marked blue) on the boundaries between adjacent bricks of differing levels of detail. Our adaptive sampling solution requires fewer sampling operations (d) and and in combination with spatiotemporal blending it results in smooth transitions across brick boundaries (c). The intensity in (b) and (d) corresponds to the number of iterations where the color white indicates a number of iterations exceeding 1024.

metrics frequently deliver large differences in level-of-detail adjacent to homogenous regions in the volume. If the working set is not restricted, then it is not obvious how to sample efficiently at decreasing density over multiple bricks and multiple levels of ancestry towards the boundary of a coarser neighbor. Limiting the transition to one brick only in this case is not advisable either, since it leaves the boundary between the bricks in question noticeable to some extent and would not provide for similarly convincing visual results such as ours.

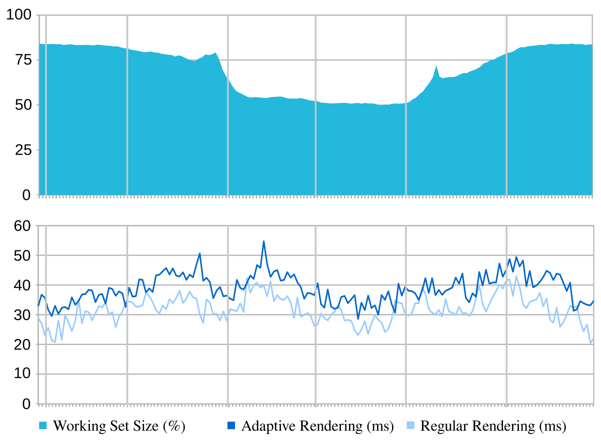
Our ray-guided working set selection strategy operates under the assumption that bricks should be refined as soon as any ray requires a higher level of detail, but a brick should be collapsed only if all rays agree that its level of detail is too high. We realize this assumption by storing the *maximum* difference in level-of-detail required by any ray to the level-of-detail actually sampled per brick in the feedback table. Alternatively, it is conceivable to store the *average* difference in level-of-detail, such that few rays requiring a high level of detail do not necessarily force a split if the majority of rays do not need a finer LOD. However, in practice, this could produce inferior working sets that do not converge to the best possible level of detail for all rays.

7 Conclusion and Future Work

Our main contribution is a ray-guided artifact-optimized volume rendering scheme and an efficient level-of-detail feedback mechanism to guide the working set selection. We generalized the work of Carmona et al. [3] from static spatial LOD blending to dynamic spatiotemporal LOD blending to provide visually pleasing transitions between adjacent bricks of differing levels of detail as well as animated transitions over time during working set adjustments. For the reduction of sampling artifacts in multi-resolution rendering, we demonstrated a consistent adaptive sampling technique that accounts for spatiotemporal blending and avoids oversampling as



(a)



(b)

Figure 7: These plots illustrate the performance of our system in a simple rotation scenario at a display resolution of 1920x1080. The seismic dataset rotates once about the y-axis as depicted in the top rows (a). The stacked area chart shows the duration of the individual algorithmic steps of the per-frame working set update in milliseconds. The memory consumption of our working set over time is depicted in the area chart in (b). The bottom graph shows the raycasting time in milliseconds for both regular state-of-the-art volume raycasting as well as our adaptive spatiotemporal rendering scheme.

well as abrupt changes of the sampling stepsize. In addition, we provide implementation details for a priority-distribution algorithm, connecting the feedback generated during raycasting with a split-and-collapse working set selection strategy. Our feedback-scheme allows occluded parts of the scene and bricks outside the frustum to collapse automatically and optimizes memory utilization in a straightforward manner.

We plan to augment our ray-guided metric with the data-based distortion suggested by Ljung et al. [10] to focus on volume regions with the highest information density. Of course, it is straightforward to use our algorithm in conjunction with quadtrees to achieve spatiotemporal LOD-blending for large image datasets. Currently, we are working on a visualization system for a series of large 3D volumes captured over time. However, it is less obvious how to include our contributions into the visualization of such 4D data sets, which already have an inherent temporal component.

Acknowledgements

We thank Rhadamés Carmona for early discussions of the spatiotemporal blending approach and Christopher Lux for providing the out-of-core volume rendering framework which we used to implement the techniques presented in this paper. This work was supported in part by the German Federal Ministry of Education and Research (BMBF) under grant 03IPT704X (project Big Data Analytics) and by the VRGeo Consortium. The seismic data set shown in this work is courtesy of Crown Minerals and the New Zealand Ministry of Economic Development (www.crownminerals.govt.nz

References

[1] J. Beyer, M. Hadwiger, and H. Pfister. State-of-the-art in gpu-based large-scale volume visualization. *Computer Graphics Forum*, 34(8):13–37, 2015.

[2] R. Carmona and B. Fröhlich. Error-controlled real-time cut updates for multi-resolution volume rendering. *Computers & Graphics*, 35(4):931–944, 2011.

[3] R. Carmona, G. Rodríguez, and B. Fröhlich. Reducing artifacts between adjacent bricks in multi-resolution volume rendering. In *Advances in Visual Computing*, volume 5875 of *Lecture Notes in Computer Science*, pages 644–655. Springer Berlin Heidelberg, 2009.

[4] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann. Gigavoxels: Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*, pages 15–22. ACM, 2009.

[5] K. Engel. Cera-tvr: A framework for interactive high-quality teravoxel volume visualization on standard pcs. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 123–124. IEEE, 2011.

[6] T. Fogal, A. Schiewe, and J. Krüger. An analysis of scalable gpu-based ray-guided volume rendering. In *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*, pages 43–51. IEEE, 2013.

[7] E. Gobbetti, F. Marton, and J. A. I. Guitián. A single-pass gpu ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*, 24(7-9):797–806, 2008.

[8] M. Hadwiger, J. Beyer, W.-K. Jeong, and H. Pfister. Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2285–2294, 2012.

[9] P. Ljung, C. Lundström, and A. Ynnerman. Multiresolution interblock interpolation in direct volume rendering. In *Proc. EUROGRAPHICS / IEEE-VGTC Symposium on Visualization and Graphics 2006*, pages 256–266, 2006.

[10] P. Ljung, C. Lundstrom, A. Ynnerman, and K. Museth. Transfer function based adaptive decompression for volume rendering of large medical data sets. In *Volume Visualization and Graphics, 2004 IEEE Symposium on*, pages 25–32. IEEE, 2004.

[11] C. Lux and B. Fröhlich. Gpu-based ray casting of multiple multi-resolution volume datasets. In *Advances in Visual Computing*, pages 104–116. Springer, 2009.

[12] H. Younesy, T. Möller, and H. Carr. Improving the quality of multi-resolution volume rendering. In *ISVC (2)*, pages 251–258. Eurographics/IEEE-VGTC Symposium on Visualization 2006, 2006.