Space-Time Volumetric Depth Images for In-Situ Visualization

Oliver Fernandes[*] University of Stuttgart
Steffen Frey[†] University of Stuttgart
Filip Sadlo[‡] University of Stuttgart
Thomas Ertl[§] University of Stuttgart

ABSTRACT

Volumetric depth images (VDI) are a view-dependent representation that combines the high quality of images with the explorability of 3D fields. By compressing the scalar data along view rays into sets of coherent supersegments, VDIs provide an efficient representation that supports a-posteriori changes of camera parameters. In this paper, we introduce space-time VDIs that achieve the data reduction that is required for efficient in-situ visualization, while still maintaining spatiotemporal flexibility. We provide an efficient space-time representation of VDI streams by exploiting inter-ray and inter-frame coherence, and introduce delta encoding for further data reduction. Our space-time VDI approach exhibits small computational overhead, is easy to integrate into existing simulation environments, and may help pave the way toward exascale computing.

Index Terms: I.3.3 [Computer Graphics]: Picture/Image Generation; I.6.6 [Simulation and Modeling]: Simulation Output Analysis

1 INTRODUCTION

Scientific computing and parallel visualization are at the cusp of the new era of exascale computing. In-situ visualization plays a key role in this transition, since the involved data rates are expected to be beyond the capabilities of future storage systems. A central difficulty with in-situ visualization in parallel computing is to conform to bandwidth constraints. On the one hand, parallel scientific computing requires very high bandwidth, e.g., for the communication between different simulation processes, on the other hand, high-resolution visualization at high frame rates is a prerequisite for successful research and development. While many simulations are intrinsically global and require substantial parallelization traffic, there is large potential for optimization in the visualization stage.

Since the available bandwidth typically does not allow for the transmission of the entire data stream to dedicated visualization nodes, the data reduction has to take place directly at the simulation nodes. In this paper, our approach is to generate intermediate visualization representations directly at the compute nodes for each time step of the simulation, and communicate those to the visualization nodes. There, they can be subsequently used to generate the final image. In particular, we focus on volume visualization due to its wide applicability throughout many areas of science and engineering. In this context, intermediate visualizations traditionally consist of color images accompanied with opacity and depth information, which are eventually composited to yield the resulting image. However, visual interaction, e.g., a camera rotation, requires the communication of changes from the visualization to the compute nodes, as well as the subsequent transfer of the visualization results back to the visualization nodes. As each frame is subject to strict constraints in bandwidth and latency, the high and continuous frame rates that are necessary to provide perceptually appropriate data exploration typically cannot be achieved this way in large-scale simulation setups.

Two main sources of visualization traffic can be distinguished: update of the simulation data, and interaction with the visualization. While visualization traffic due to update of simulation data is inevitable, it is one of the aims of this paper to avoid or at least reduce visualization traffic due to interaction. We decouple these two processes by maintaining an explorable space-time representation of the volume rendering on the visualization nodes. This approach not only avoids traffic due to visual interaction, it also allows for efficient temporal exploration, i.e., investigating previous data while the simulation continues. Critically, this decoupled exploration maintains full interactivity even when data updates are rare in case of slow simulation.

Our approach is based on volumetric depth images (VDI) [4], which have been recently proposed as an extension of layered depth images (LDI) [18] for volume data. LDIs are a view-dependent representation with multiple 'pixels' along each line of sight, that allows for deferred interaction. Each of these 'pixels' features a distinct depth value to represent the intersection of the viewing ray with geometry. In contrast, VDIs represent a generalization for volumes and store depth intervals instead. VDIs provide a dense representation of the volume that can be rendered with arbitrary camera configurations. VDIs represent classified volume data, i.e., color and opacity after application of a transfer function. We extend VDIs to a space-time representation, with the goal to yield better compression and achieve the temporal decoupling of simulation and visualization.

In particular, our contributions include:

- Space-time VDI representation for decoupled interactive in-situ volume visualization.
- Parallel generation, compression, and reconstruction of space-time VDIs inducing only small overhead in parallel simulation environments.

2 RELATED WORK

In-Situ and Remote Rendering

Remote rendering based on the transmission of rendered frames has been used for a wide range of applications, including rendering on mobile devices [1], cloud gaming (OnLive or Gaikai), protection of proprietary data [6], as well as in-situ visualization of time-varying volume data [11] and visualization in medical applications [3]. To maintain interactivity, the transfer size is typically reduced by employing JPEG [6] or MPEG [5] compression. An alternative technique employing a CUDA-based parallel compression method was presented by Lietsch and Marquardt [9], while Pajak et al. [14] discuss efficient compression and streaming of frames rendered from a dynamic 3D model. Level-of-detail techniques are frequently employed [13, 6] to handle excessive server load. However, in these approaches, the chosen level of detail has no direct influence on the image data size that needs to be transferred to the display client.

Explorable Images

Shade et al. [18] introduced LDIs representing one camera view with multiple pixels along each line of sight. Multi-layered representations have since been popularized in commercial rendering

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

IEEE Symposium on Large Data Analysis and Visualization 2014
October 9–10, Paris, France
978-1-4799-5215-1/14/$31.00 ©2014 IEEE

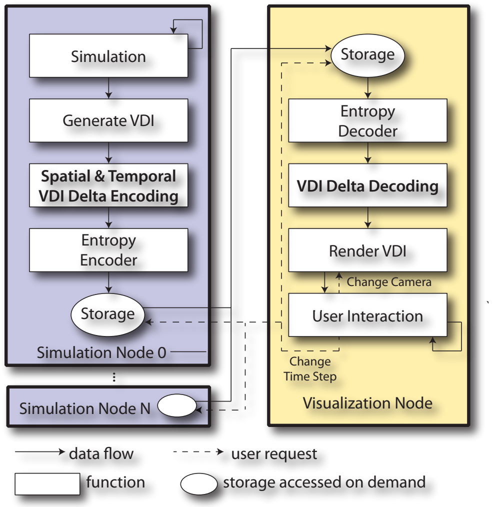


Figure 1: Integrated simulation and visualization system.

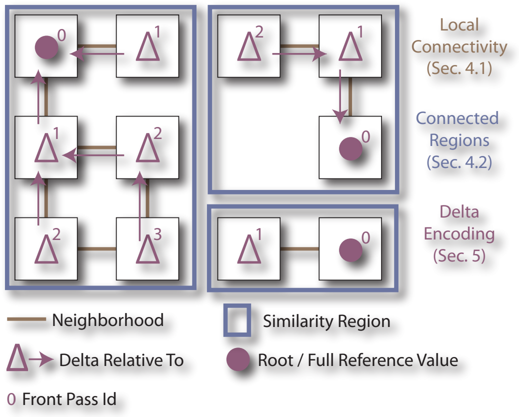

Local Connectivity (Sec. 4.1)
Connected Regions (Sec. 4.2)
Delta Encoding (Sec. 5)
Neighborhood
Similarity Region
Delta Relative To
Root / Full Reference Value
0 Front Pass Id

Figure 2: The stages of space-time VDI generation.

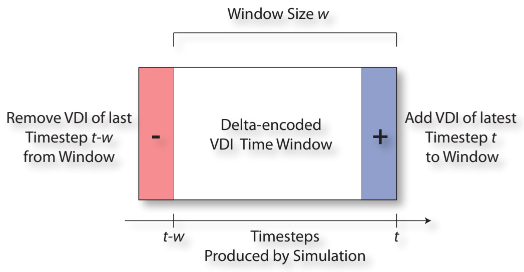

Window Size w
Remove VDI of last Timestep t-w from Window
-
Delta-encoded VDI Time Window
+
Add VDI of latest Timestep t to Window
t-w
Timesteps Produced by Simulation
t

Figure 3: Sliding window approach to constantly update our VDI representation with new simulation results.

software to simulate complex materials like skin on synthetic objects [2]. In volume rendering, layer-based representations have been used to defer operations such as lighting and classification [17, 16]. Layer-based representations have also been proven effective to cache results [10, 8] or certain volumetric properties along the view rays [12], which can be reused for efficient transfer function exploration. Tikhonova et al. [20] convert a small number of volume renderings to a multi-layered image representation, enabling interactive exploration in transfer function space. In another work, Tikhonova et al. [21] use an intermediate volume data representation based on ray attenuation functions, which encode the distribution of samples along each ray. Shareef et al. [19] use image-based modeling to allow for efficient GPU-based rendering of unstructured grids based on parallel sampling rays and 2D texture slicing. In contrast to VDIs which represent volumes, LDIs and related techniques are targeted toward surfaces (with specific depth values and 'vacuum' in between).

Volumetric Depth Images

Volumetric depth images are a condensed representation for classified volume data, providing high quality and reducing both render time and data size considerably. Instead of only saving one color value for each view ray as in standard images, or sets of pixels as in the case of LDIs, VDIs store a set of so-called supersegments, each consisting of a depth range with composited color and opacity. This compact representation is independent from the representation of the original data and can quickly be generated by a slight modification of existing raycaster codes. VDIs can be rendered efficiently at high quality with arbitrary camera configurations [4].

VDIs can be generated efficiently during volumetric raycasting from a certain camera configuration by partitioning the samples along view rays according to the similarity of the composited color, providing the additional possibility to skip 'empty' regions. These partitions are then stored as lists of so-called supersegments containing the bounding depth pair (z_f, z_b) and respective accumulated color and opacity value. Geometrically, each (quadrilateral) pixel in the image plane forms a pyramid. For visualization, the supersegments are rendered as frustums of these pyramids. Frustums generated by the same ray are conceptually grouped into a frustum list. Color and opacity are modeled constant within a frustum, as determined during VDI generation. During rendering, the frustum lists are depth-ordered and composited. For this, the length l a ray passes through each frustum needs to be accounted for, in order to compute its opacity contribution. Note that VDIs provide renderings identical to those of the original data when rendered from the view used for their generation, but provide high-quality approximations also for deviating views.

3 Overview

This paper addresses typical in-situ visualization setups consisting of a parallel simulation environment with integrated visualization (Fig. 1). $N+1$ simulation nodes generate the data, employing domain decomposition for parallel execution. On each compute node, we generate a VDI from each time step of the respective part of the simulation domain. To achieve the compression rates required for in-situ visualization and to provide replay functionality for exploring previous time steps, we generate a space-time VDI from the temporal sequence of traditional VDIs. In detail, we extend the clustering process—that originally only operates along rays—across view rays and time steps. Our spatiotemporal VDI representation is continuously updated with new time steps, i.e., new VDIs from the simulation are continuously incorporated.

Fig. 2 gives an overview on the main steps of our algorithm for space-time VDI delta encoding, i.e., on removing statistical redundancy from the space-time VDI representation. We first establish groups of space-time connected supersegments (Sec. 4). Then, based on this, we employ delta encoding for further data reduction (Sec. 5). To limit memory requirements, we use a sliding-window approach in time for our space-time VDI construction (Fig. 3), i.e., as soon as the window reaches its maximum capacity, the oldest time step VDI is removed from the space-time representation.

Finally, the compressed data is stored locally. Upon request, it is transfered to the client, i.e., to the visualization node, or some intermediate storage, where it is cached locally. Our implementation uses a combination of Huffman and LZ4 compression for fast

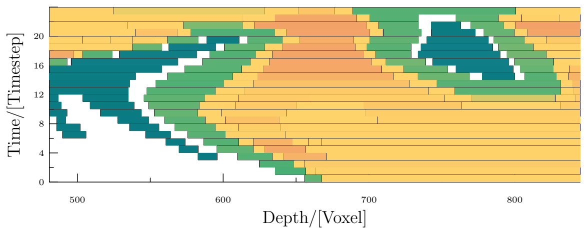


Figure 4: Illustration of a space-time VDI excerpt.

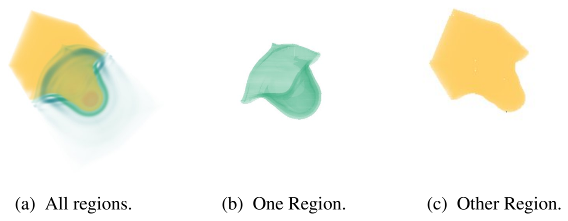


Figure 5: Connected regions determined by our algorithm.

and efficient encoding. On the visualization node, the pipeline is executed in reverse order (Fig. 1), providing a sequence of VDIs that are rendered efficiently using opacity-corrected blending and rasterization [4]. This not only allows for interactive exploration by the user, it also provides temporal navigation within the time interval spanned by the sliding window. Among others, this may also be employed to enable computational steering of the simulation (e.g., [7, 15]).

4 Establishing Connectivity

A space-time VDI can be seen as a collection of supersegments, as illustrated schematically in Fig. 4. In order to reduce the size of the VDI sequence within the sliding time window, i.e., to merge supersegments to coherent regions, we first determine the connectivity of the supersegments in terms of spatiotemporal adjacency (*Local Connectivity*, Sec. 4.1). In detail, this means that we determine for each supersegment v its space-time neighborhood $N(v)$. Each coherent region is represented by an id and thus at initialization, when the sliding window contains only a single VDI, we assign a unique id to each supersegment. Next, from the local connectivity information and based on color similarity, regions of connected components are determined by propagating the ids between connected supersegments, also to those of VDI time steps newly inserted into the sliding time window. To assure global merging of the regions, we add a subsequent region labeling pass (*Region Labeling*, Sec. 4.2), resulting in the connected regions of supersegments (Fig. 5).

4.1 Local Connectivity

The local connectivity between supersegments is determined in two passes, applying two different criteria, one in space-time and one in terms of color similarity. Space-time connectivity between two supersegments v, w is given iff

1. they were generated from adjacent pixels (according to 4-connectivity) in the same time step *or* they belong to the same pixel in successive time steps, and if
2. their depth ranges z_v and z_w overlap.

This first pass establishes a spatiotemporal graph G, where connectivity represents edges and the supersegments represent nodes.

In the second pass (Alg. 1), we propagate the ids along G but not across edges where, similar to the merging criterion along rays

Algorithm 1 Local connectivity computation. Region ids are propagated along edges $V(G)$ of the space-time connectivity graph G.

```
procedure ColorSimilarity
    // Add all supersegments to the current changeset $C$
    $C \leftarrow V(G)$
    repeat
        $D \leftarrow \emptyset$
        for all $v \in C$ do
            for all $w \in N(v)$ do
                // Compare supersegment color to
                // neighboring region color
                if $i(v) > i(w) \wedge d(c(v), c(i(w))) < \varepsilon$ then
                    $i(v) \leftarrow i(w)$
                    // Add all neighbors
                    $D \leftarrow D \cup N(v)$
        $C \leftarrow D$
    until $D = \emptyset$
```

Algorithm 2 Region labeling. Determine connected regions along edges $V(G)$ of the space-time connectivity graph G.

```
procedure ComponentLabeling
    // Add all supersegments to the current changeset $C$
    $C \leftarrow V(G)$
    repeat
        $D \leftarrow \emptyset$
        for all $v \in C$ do
            for all $w \in N(v)$ do
                if $i(v) > i(w)$ then
                    $i(v) \leftarrow i(w)$
                    $D \leftarrow D \cup N(v)$
        $C \leftarrow D$
    until $D = \emptyset$
```

during VDI generation [4], the Euclidean distance $d(\cdot,\cdot)$ of premultiplied color in RGB space exceeds a user-defined threshold ε (Alg. 1, Line 10). We denote by $i(v) \leftarrow i(w)$ the propagation of id $i(w)$ of supersegment w to supersegment v, and propagate the smallest id if multiple edges apply. Note that $c(v)$ represents the original color of supersegment v while $c(i(v))$ represents the color of the coherent region with id $i(v)$. Coherent regions inherit the original color of the supersegment where the id was propagated from, hence the color distance between a supersegment and the color of its containing coherent region is constrained by ε. Alg. 1 is executed whenever a new VDI time step is added to the sliding window, after assigning each of this VDI's supersegment a unique id, and after updating G according to spatiotemporal connectivity. Note however, that, due to disruption of regions during propagation, it is still possible for regions to be spatiotemporally disconnected but to share the same region id.

4.2 Region Labeling

To ensure unique ids for the regions obtained so far, we employ connected-component labeling (Alg. 2). The key difference between Alg. 1 and Alg. 2 is that the color test can now be omitted, since color similarity has already been confirmed. Finally, we generate a set of root supersegments r by simply selecting one supersegment from each region. As explained in detail in Sec. 5.1, these root supersegments are employed to convert the individual region graphs for delta encoding.

5 Delta computation

To reduce the overall size of the space-time VDI representation, the data structure containing the geometry of the coherent regions is

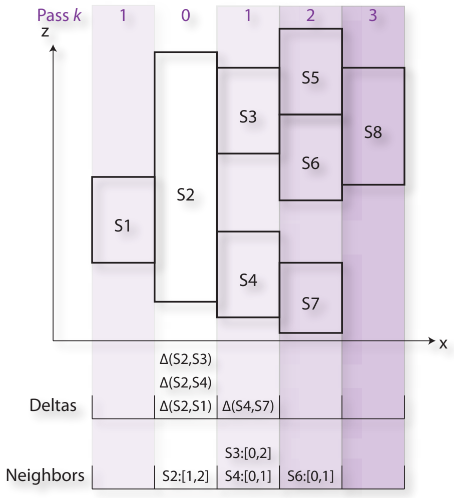


Figure 6: Delta encoding of a region with supersegment S2 being the root r. The x-axis is in the pixel domain, while the z-axis denotes depth.

Algorithm 3 Delta Encoding ($\otimes$ denotes concatenation.)

```
1:  procedure DeltaEncoding
2:      $F_0 \leftarrow p(r)$
3:      $c_{acc} \leftarrow 0$
4:      $\Sigma_G \leftarrow \emptyset$
5:      repeat
6:          $F_{k+1} \leftarrow \emptyset$
7:          // Gather new frontier and determine its size
8:          for all $p \in F_k$ do
9:              $a_p \leftarrow 0$
10:             for all $v \in p$ do
11:                 if marked$(v) \wedge \neg$visited$(v)$ then
12:                     visited$(v) \leftarrow$ true
13:                     for all $w \in N(v)$ do
14:                         $a_p \leftarrow a_p + 1$
15:                         marked$(w) \leftarrow$ true
16:                         $F_{k+1} \leftarrow F_{k+1} \cup p(w)$
17:         $F_k \leftarrow$ Sort$(F_k)$
18:         $s_p \leftarrow$ PrefixSum$(a_{p \in F_k})$
19:         $\sigma \leftarrow \emptyset$
20:         // Output deltas and neighbor counts from new frontier
21:         for all $p \in F_{k+1}$ do
22:             for all $w \in p$ do
23:                 if marked$(w)$ then
24:                     $c_{acc} \leftarrow c_{acc} + c(w)/|z|$
25:                     $\Delta \leftarrow z(w) - z(v)$
26:                     $\sigma_{s_p} \leftarrow \Delta \cup a_p$
27:         $\Sigma_G \leftarrow \Sigma_G \otimes \sigma$
28:         $k \leftarrow k + 1$
29:     until $F_{k+1} = \emptyset$
30:     // Store normalized region color
31:     $c_G \leftarrow c_{acc}$
```

reorganized and prepared for efficient compression by the entropy encoding scheme. Based on the fact that field data is typically continuous, and the assumption that data changes sufficiently linearly on a small scale (i.e., in the order of several units of the underlying sampling grid), entropy encoding can achieve better compression if merely delta differentials need to be stored (Sec. 5.1). To reconstruct the original values, i.e., to decode the data, one must integrate along the path along which the differentials were obtained, starting at the integration boundary (Sec. 5.2).

5.1 Delta Encoding

For each region, we define paths in our connectivity graph G for delta encoding, by traversing it in a breadth-first manner, thereby establishing a tree structure. This conversion of the graph G to a tree is parallelized by exploiting the fact that supersegments are already organized in a certain manner from the VDI generation process, i.e., groups of supersegments generated from the same ray belong to the same pixel. As a consequence, spatially adjacent supersegments necessarily belong to neighboring pixels. The pixels $p(v)$ belonging to every supersegment v (node of G) residing at tree depth k are collected in the frontier set $F_k(G)$ (Fig. 6). Initially, we start at the root r of each region graph G that has been determined in the region labeling step (Sec. 4.2). The respective supersegment is marked as a frontier node, and its corresponding pixel is added to the frontier set $F_0(G)$.

In every iteration k, our delta encoding algorithm first checks the current frontier nodes to gather their neighbors as the new frontier (Alg. 3, Lines 8–16). For each pixel p in the set $F_k(G)$, we check for marked frontier nodes. If a marked frontier node v is found, its mark is removed and its status is changed to 'visited'. We traverse the node edge list in a fixed order, marking all connected neighbors w, while omitting the visited ones, and storing their pixels $p(w)$ in $F_{k+1}(G)$. The number of marked neighbors is stored in a_p. Once the new frontier $F_{k+1}(G)$ has been gathered, its elements are sorted by pixel position in the image, and a prefix scan on the neighbor counts a_p defines storage offsets s_p for all pixels in the current frontier. This enables us to process the elements of $F_{k+1}(G)$ in parallel, while maintaining a well-defined sequence for remapping the data to the respective pixels when reconstructing the region later on.

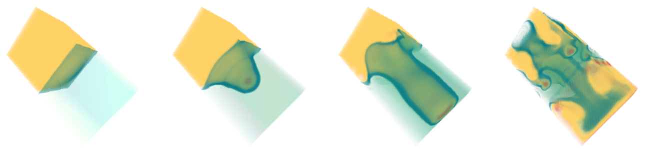
In a second step, the newly marked neighbors and the storage offsets s_p are used to convert their data into a stream (Alg. 3, Lines 21–27). Instead of storing the topology directly, we simply save the neighbor count for every direction. Since the edge list was traversed in a well-defined order, the mapping for our data stream is unambiguous, for each pixel. For every neighbor w of every segment v (from all pixels collected in $F_k(G)$), the difference of their depth range z is stored, and its color $c(w)$ is accumulated in c_{acc}, weighted by the supersegment length $|z|$. This results in a unique set of deltas, and a neighbor count set (for all directions) for every node v in pixel p, from which all neighbors can later be reconstructed by means of their relative position. With the prefix scan providing a defined stream position for all individual pixels and the cumulative substream from all pixels p can be turned into a byte sequence σ which is added to the region output stream Σ_G, and the algorithm iterates on the set $F_{k+1}(G)$. Once $F_{k+1}(G)$ is empty, the region's final color c_G is stored along with the position for the root r.

5.2 Delta Decoding

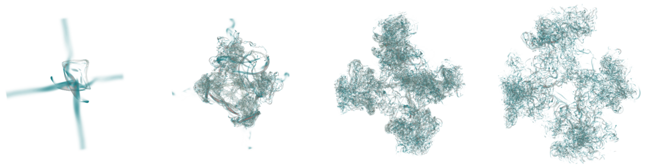
For interactive visualization on the client side, a VDI needs to be reconstructed from the stream generated in Sec. 5.1. To match the construction of the tree from Sec. 5.1, the algorithm rebuilds the data in a corresponding breadth-first fashion (Alg. 4). For every region, the root node is added to the VDI in construction at the position stored by (Alg. 3). The pixel containing the root is added to the frontier set F_0. The algorithm then builds the region by retrieving the data for each tree depth from the stream to construct the next set of connections. The frontier F_k is then sorted by pixel position. This ensures that the pixels have the same sequence within the current frontier as during the corresponding encoding iteration k. For every node in the pixels in F_k, a neighbor count is retrieved from the stream, which determines the exact number of neighbors for each direction, due to the fixed order defined earlier. The cor-

Algorithm 4 Reconstruction of a single region.

```
 1: procedure RECONSTRUCT($S$)
 2:     // Add pixel containing $r$ from Stream $\Sigma_G$
 3:     // Store the reconstruction in result $R$
 4:     $F_0 \leftarrow p(r)$
 5:     $R \leftarrow \emptyset$
 6:     repeat
 7:         $F_k \leftarrow \text{Sort}(F_k)$
 8:         $R \leftarrow \text{Pop}(\Sigma_G)$
 9:         $F_{k+1} \leftarrow \emptyset$
10:         for all $p \in F_k(G)$ do
11:             for all $v \in p$ do
12:                 if marked($v$) $\wedge \neg$visited($v$) then
13:                     visited($v$) $\leftarrow$ true
14:                     $R \leftarrow R \cup \Delta$
15:                     marked($\Delta$) $\leftarrow$ true
16:                     $F_{k+1}(G) \leftarrow F_{k+1}(G) \cup p(\Delta)$
17:         $k \leftarrow k+1$
18:     until $F_{k+1}(G) = \emptyset$
```



(a) Rayleigh-Taylor dataset, $128 \times 128 \times 256$, 24 time steps (2, 7, 12, and 17).



(b) λ_2 dataset, $529 \times 529 \times 529$, 77 time steps (0, 20, 40, and 60).

Figure 7: Renderings for different time steps of our datasets.

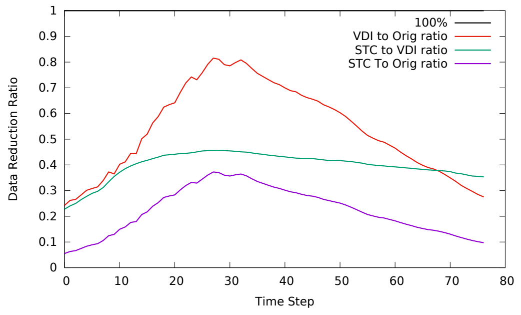
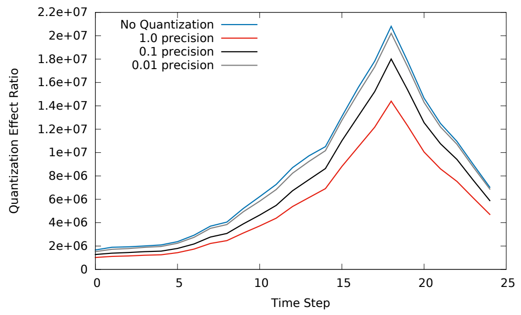
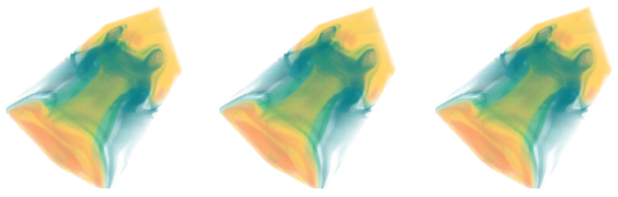


Figure 8: Compression ratios for the λ_2 dataset.



(a) Compression ratio



(b) None (c) q_g=0.1 (d) q_g=0.01

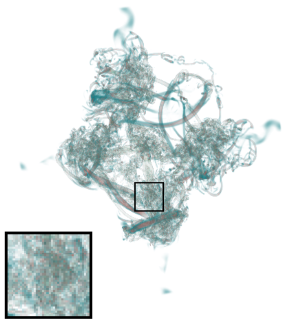
Figure 9: Different geometrical quantization q_g settings for the Rayleigh-Taylor dataset using original colors.

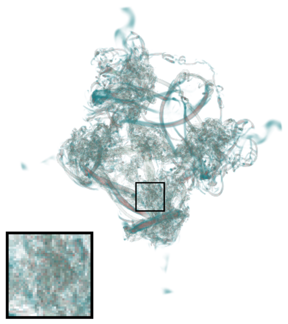
rect number of deltas necessary to construct all neighbors is now known as well and can be retrieved from the stream. The neighbors of a node v are then constructed by adding the delta to the current nodes depth range value z. Also, the containing pixel of any newly constructed node is added to $F_{k+1}(G)$. The algorithm iterates until $F_{k+1}(G)$ evaluates empty.

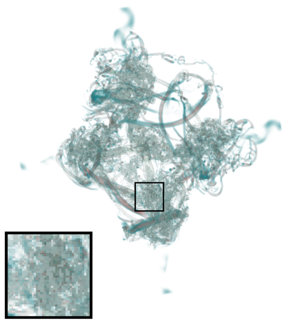
6 RESULTS

For our performance measurements, we analyze the results of two time-dependent datasets from simulation in detail: the Rayleigh-Taylor dataset and the λ_2 dataset (Fig. 7). The Rayleigh-Taylor dataset shows two fluids of different density under the influence of gravity. It consists of 24 individual timesteps, each having the dimensions $128 \times 128 \times 256$, and a size of 8.0 MB. The λ_2 dataset depicts the temporal development of a vortex cascade, visualized with the λ_2 criterion. It consists of 76 individual timesteps, with the dimensions $529 \times 529 \times 529$, and a size of 142.0 MB each. The data is defined on a grid of uniform voxels. As our method is based on the representation generated by the VDI step, compression ratios are presented by comparing VDI output to our space-time clustering (STC). As the Rayleigh-Taylor dataset has larger regions of similar color, it fullfills the assumption of a locally linearly changing dataset a better than the λ_2 data, and has larger regions of similar color for the clustering. The λ_2 dataset is rather noisy and produces a VDI with many smaller regions. The sizes of these datasets represent the chunks of a large-scale simulation, which are available on a single node at a given time. Unless noted otherwise, data size comparisons are given relative to output compressed by a lossless entropy encoder. To better exploit similar values for depth differences, accuracy was reduced by truncating floating point values to a limited amount of decimals. The remaining accuracy is denoted as q_g. For VDIs, geometrical quantization has been applied with the same precision as for STC, to keep results comparable. Unless noted otherwise, we used a color merge ε value of 0.001, a geometry quantization precision q_g of 0.1, and an image resolution of 512×512 throughout our measurements. The performance measurements were obtained on a node featuring a 3.4 GHz CPU with 8 cores and a GeForce GTX580. While the VDI is generated on the GPU using CUDA, our STC component runs in parallel on the CPU using OpenMP.

Fig. 8 shows the compression ratios that were obtained with the λ_2 dataset. It can be seen that significant reductions in data size can be achieved, with only marginal decrease in rendering quality, as discussed in detail below. Despite the high variety in the data, VDIs already achieve a reduction down to 25%–40% in comparison to the classified volume size. Our STC is able to further reduce this significantly down to 10%–30%. The ratio between VDIs and STC is relatively constant. However, in particular in the beginning with larger, more homogeneous regions in the dataset, the strongest compression ratios are achieved, as can be expected.


(a) Original VDI, 19.4 kB


(b) STC $\varepsilon = 0.0001$, 13.3 kB


(c) STC $\varepsilon = 0.001$, 13.0 kB

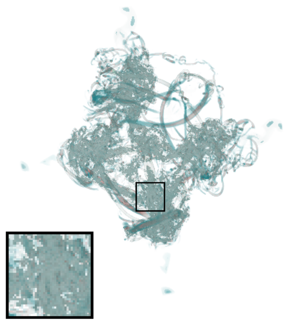
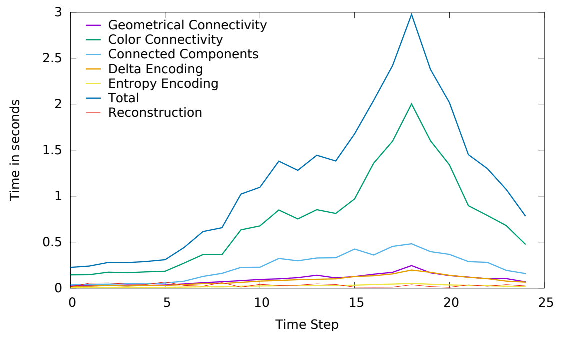
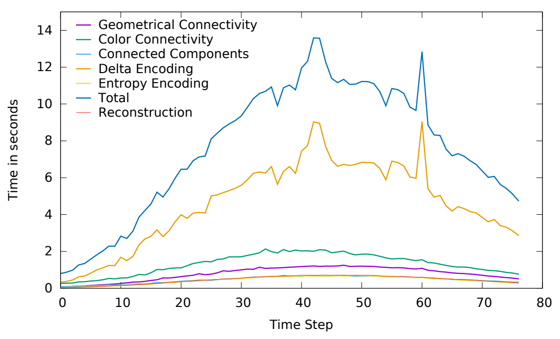

(d) STC $\varepsilon = 0.01$, 11.0 kB

Figure 10: Different color merge settings applied to the λ_2 dataset.



(a) Rayleigh-Taylor dataset.



(b) λ_2 dataset.

Figure 11: Timings across different time steps for the Rayleigh-Taylor and the λ_2 dataset.

Table 1: Color mean square errors for the Rayleigh-Taylor dataset. The compression ratio is given for color data only in comparison to original VDI.

User error	MSE	Max	Comp. ratio
0.1	0.0110	0.0127	0.07%
0.01	0.00192	0.002 26	1.71%
0.001	0.00024	0.000 27	36.06%

Fig. 9 shows the influence of different geometric quantization settings q_g. Stronger geometry quantization leads to a higher compression, yet trading this reduction in size for a loss in quality. However, for the λ_2 dataset, the total data size is influenced more strongly by the color values. Fig. 10 depicts the impact of different color merge values ε. It can be seen that the larger the ε, better compression can be achieved, yet at the cost of loss of detail.

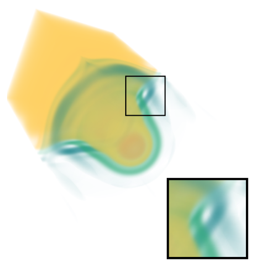
Fig. 11 shows the timings of different steps of our STC approach for the Rayleigh-Taylor (Fig. 11(a)) and the λ_2 dataset (Fig. 11(b)). In general, compared to the delta encoding step, the local connectivity computation takes considerably longer and dominates the total run time. It can also be seen that timings differ significantly with datasets and time steps, i.e., the complexity of the underlying data. It can also be seen that reconstruction is a very fast process, taking significantly less than a second in any of our experiments. Fig. 13 shows renderings of different views. It can be seen that our technique is able to sustain good quality for a variety of view perspectives, while also reducing data size significantly as demonstrated above.

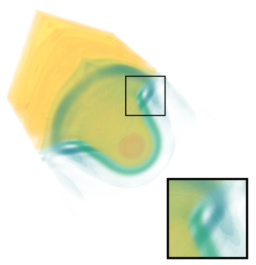
To get a comparison for the loss of quality, we matched the actual mean square error in color distance to the user defined limit ε for this error. The results are shown in Tab. 1. The table shows the average error for all timesteps, and largest deviation. For direct visual comparison, see Fig. 12. Note how the premultiplied color fails for large ε, and introduces heavy artifacts, without any significant gain to compression.

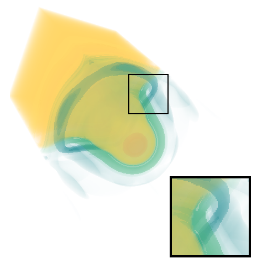
Discussion and Limitations. As basically all in-situ approaches, the STC method loses information that is outside the defined scope of interest. In our case, any volume data not inside the viewport during VDI generation is discarded and cannot be reconstructed. Some prior information regarding the data is also required for choosing an appropriate transfer function. This also has an influence on the compression ratio that can be achieved. Additionally, the merging criterion used here (Euclidian premultiplied color distance) may lead to artifacts in regions of low opacity.

7 Conclusion

In this paper, we extended VDIs to exploit space-time coherence typically found in time-dependent scientific data, with the main goal being the efficient representation of volumes stemming from large-scale simulations. The coherence and continuity between time steps can be efficiently exploited to achieve an in-situ data reduction, which caters to the bandwidth-critical nature of exascale settings. For future work, we intend to test the system in the context of a real-world large-scale in-situ simulation setup. In addition, our algorithms were designed with massive parallelism in mind, but only evaluated on multicore CPUs in the context of this paper. We therefore intend to evaluate the usage of GPUs with the goal to further diminish the processing overhead.


(a) Raycast


(b) $\varepsilon = 0.001$


(c) $\varepsilon = 0.01$

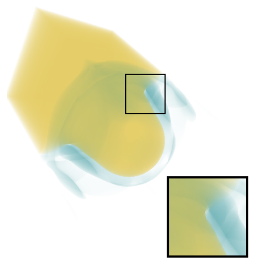
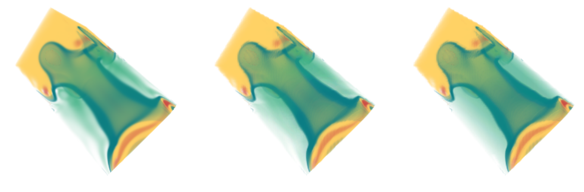
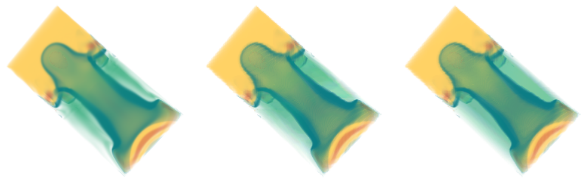

(d) $\varepsilon = 0.1$

Figure 12: Rayleigh-Taylor dataset space-time clusterings merged for different color error settings.


(a) 0° (Raycasting, VDI Original, STC)


(b) 20° (Raycasting, VDI Original, STC)

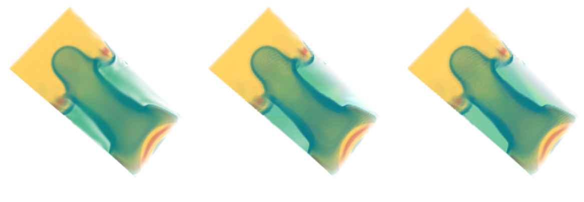

(c) 40° (Raycasting, VDI Original, STC)

Figure 13: Renderings of time step 14 of the Rayleigh-Taylor dataset at different angles, comparing raycasting, VDI, and STC.

Acknowledgements

This work was primarily funded by Deutsche Forschungsgemeinschaft (DFG) under grant SPP 1648 (ExaScale FSA) and the Cluster of Excellence in Simulation Technology (EXC 310/1). The authors would like to thank Andrea Beck (IAG, University of Stuttgart) for providing the vortex cascade dataset and Verena Krupp (Simulation Techniques & Scientific Computing, University of Siegen) for providing the Rayleigh-Taylor instability dataset.

References

[1] J. Diepstraten, M. Gorke, and T. Ertl. Remote line rendering for mobile devices. In *Computer Graphics International*, pages 454–461, 2004.

[2] C. Donner and H. W. Jensen. Light diffusion in multi-layered translucent materials. *ACM Trans. Graph.*, 24(3):1032–1039, 2005.

[3] K. Engel, T. Ertl, P. Hastreiter, B. Tomandl, and K. Eberhardt. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *Proceedings of IEEE Visualization '00*, pages 449–452, 2000.

[4] S. Frey, F. Sadlo, and T. Ertl. Explorable volumetric depth images from raycasting. In *Proceedings of the Conference on Graphics, Patterns and Images*, pages 123–130, Aug 2013.

[5] R. Herzog, S. Kinuwaki, K. Myszkowski, and H.-P. Seidel. Render2MPEG: a perception-based framework towards integrating rendering and video compression. *Computer Graphics Forum*, 27(2):183–192, 2008.

[6] D. Koller, M. Turitzin, M. Levoy, M. Tarini, G. Croccia, P. Cignoni, and R. Scopigno. Protected interactive 3d graphics via remote rendering. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 695–703, 2004.

[7] O. Kreylos, A. M. Tesdall, B. Hamann, J. K. Hunter, and K. I. Joy. Interactive visualization and steering of cfd simulations. In *Proceedings of the Symposium on Data Visualisation*, pages 25–34, 2002.

[8] E. LaMar and V. Pascucci. A multi-layered image cache for scientific visualization. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 61–67, 2003.

[9] S. Lietsch and O. Marquardt. A CUDA-supported approach to remote rendering. In *Proceedings of the 3rd international conference on Advances in visual computing - Volume Part I*, ISVC'07, pages 724–733. Springer-Verlag, 2007.

[10] E. J. Luke and C. D. Hansen. Semotus visum: a flexible remote visualization framework. In *Proceedings of IEEE Visualization '02*, pages 61–68, 2002.

[11] K.-L. Ma and D. M. Camp. High performance visualization of time-varying volume data over a wide-area network. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, 2000.

[12] K.-L. Ma, M. F. Cohen, and J. S. Painter. Volume seeds: A volume exploration technique. *The Journal of Visualization and Computer Animation*, 2(4):135–140, 1991.

[13] K. Moreland, D. Lepage, D. Koller, and G. Humphreys. Remote rendering for ultrascale data. *Journal of Physics: Conference Series*, 125(1):012096, 2008.

[14] D. Pajak, R. Herzog, E. Eisemann, K. Myszkowski, and H.-P. Seidel. Scalable remote rendering with depth and motion-flow augmented streaming. *Computer Graphics Forum*, 30(2):415–424, 2011.

[15] S. Rathmayer. A tool for on-line visualization and interactive steering of parallel HPC applications. In *Proceedings of the 11th International Symposium on Parallel Processing*, pages 181–186, 1997.

[16] P. Rautek, S. Bruckner, and E. Gröller. Semantic layers for illustrative volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1336–1343, 2007.

[17] T. Ropinski, J. Prassni, F. Steinicke, and K. Hinrichs. Stroke-based transfer function design. In *IEEE/EG International Symposium on Volume and Point-Based Graphics*, pages 41–48, 2008.

[18] J. Shade, S. Gortler, L.-W. He, and R. Szeliski. Layered depth images. Proceedings on Computer Graphics and Interactive Techniques, pages 231–242, 1998.

[19] N. Shareef, T.-Y. Lee, H.-W. Shen, and K. Mueller. An image-based modeling approach to GPU-based unstructured grid volume rendering. *Proceedings of Volume Graphics*, pages 31–38, 2006.

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

[21] A. Tikhonova, C. D. Correa, and K.-L. Ma. An exploratory technique for coherent visualization of time-varying volume data. In *Proceedings of the Eurographics conference on Visualization*, pages 783–792, 2010.