# Visual Analysis of Large Multi-Field AMR Data on GPUs Using Interactive Volume Lines

Stefan Zellmann* University of Cologne
Serkan Demirci† Bilkent University
Uğur Güdükbay‡ Bilkent University

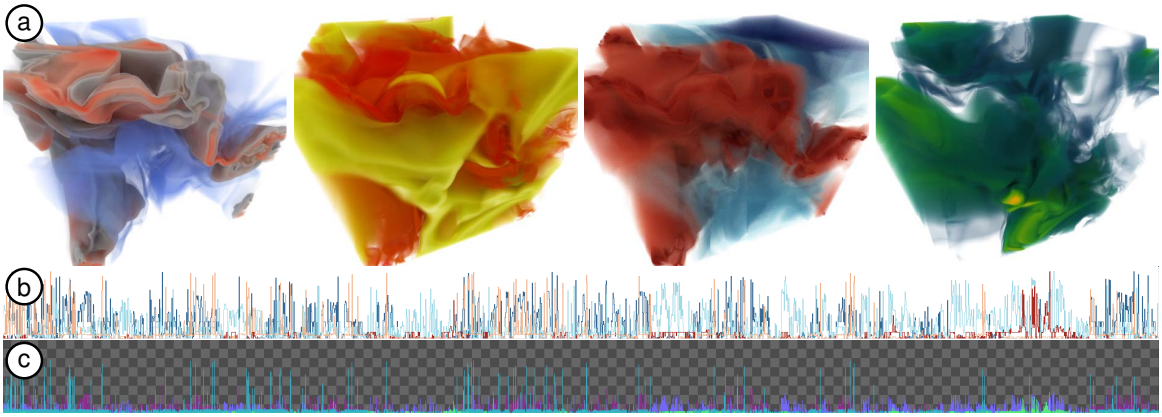


Figure 1: Coupled *interactive volume lines* (IVL) and large volume visualization. (a) shows 3D renderings of four fields of an astrophysical simulation (from left to right: density, temperature, pressure, and velocity magnitude). Our prototype interactively links such large-scale AMR data with 1D volume line plots (polyline plot representation in (b), and bar plot representation in (c)).

## Abstract

To visually compare ensembles of volumes, *dynamic volume lines* (DVLs) represent each ensemble member as a 1D polyline. To compute these, the volume cells are sorted on a space-filling curve and scaled by the ensemble's local variation. The resulting 1D plot can augment or serve as an alternative to a 3D volume visualization free of visual clutter and occlusion. Interactively computing DVLs is challenging when the data is large, and the volume grid is not structured/regular, as is often the case with computational fluid dynamics simulations. We extend DVLs to support large-scale, multi-field adaptive mesh refinement (AMR) data that can be explored interactively. Our GPU-based system updates the DVL representation whenever the data or the alpha transfer function changes. We demonstrate and evaluate our interactive prototype using large AMR volumes from astrophysics simulations.

**Index Terms:** Human-centered computing—Visualization—Visualization application domains—Visual analytics; Human-centered computing—Visualization—Visualization application domains—Scientific visualization

## 1 Introduction

We propose an interactive implementation of Weissenböck *et al.*'s [13] *dynamic volume lines* (DVLs). DVLs visualize ensemble volumes in 1D as a set of polylines. While ensemble or multi-field volume rendering may suffer from visual clutter and self-occlusion, DVLs present a viable alternative for visual exploration or can augment an existing 3D volume visualization. A fundamental problem with DVLs and similar plots is that many cells of the volume map to only a few pixels of the output viewport, leading to overdraw. The number of cells can be several orders of magnitude higher than the number of pixels that the line segments of the polylines project to, resulting in a linear mapping of cells to pixels that compresses regions where the data is not as interesting.

Weissenböck *et al.* concentrate on the visual analytics aspects of DVLs and have proven their efficacy to this end, yet the authors' work focused on smaller structured-regular volumetric data sets (on the order of $64^3$ cells). When scaling to larger volume sizes and unstructured or hierarchical grid types, interactively computing DVLs becomes a challenge we address in this paper.

Another aspect that remains unexplored by Weissenböck *et al.*'s work is that the spatial arrangement (and hence the local variation) of the volume ensemble changes when the alpha transfer function of an ensemble member is updated. However, interactive transfer function updates are an essential aspect of scientific volume visualization, and the data structures and algorithms involved in computing and updating DVLs must be carefully chosen not to prohibit this type of interaction.

We, therefore, concentrate on the performance and interactivity aspects of computing dynamic volume lines on the GPU. More specifically, we contribute

- an extension of dynamic volume lines for AMR volumes where the cell size depends on the refinement level,
- a GPU implementation that allows to interactively update the volume lines in the presence of user-editable transfer functions per ensemble member, and
- an application that allows interaction with the 3D view and the 1D plot through brushing and linking.

An overview of the visualizations our system supports is given in Fig. 1 (here exemplified using a multi-field data set).

## 2 Background and Related Work

This section reviews related works on large-scale volume visualization and adaptive mesh refinement (AMR) data. Additionally, we provide a background summary of the dynamic volume lines method by Weissenböck *et al.* [13] as our main related work on the visual analytics side.

*e-mail: zellmann@uni-koeln.de
†e-mail: serkan.demirci@bilkent.edu.tr
‡e-mail: gudukbay@cs.bilkent.edu.tr

2023 IEEE Visualization and Visual Analytics (VIS) | 979-8-3503-2557-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/VIS54172.2023.00020

979-8-3503-2557-7/23/$31.00 ©2023 IEEE



DOI 10.1109/VIS54172.2023.00020

### 2.1 Large-Scale Volume Data

Our paper concentrates on large volume data. While our prototype does include a 3D rendering component, in this section, we focus on data representation more than on the rendering side.

Although large structured volumes are still commonplace in some areas [5], unstructured or hierarchical representations are ubiquitous in the computational sciences. While unstructured meshes [8, 9] are pretty standard, many codes use *adaptive mesh refinement* (AMR) [3] to concentrate the computation on the relevant regions in space. The resulting data can be block-structured, overlapping grids, Octrees, or similar hierarchies.

Recent challenges with AMR visualization include smooth interpolation in 3D [11], GPU acceleration structures [12], and time-dependent data [16]. A common approach for representing AMR data is the one adopted by Wald *et al*. [12], where AMR cells "snap" to the *logical grid*; that hypothetical uniform grid has a resolution that when resampling the volume, the finest AMR cells occupy exactly one logical cell. Each AMR cell is unambiguously defined by its lower corner on the logical grid, and its refinement level $L$. By defining 0 to denote the finest level, the cell size can be computed as $C_w = 2^L$. This representation omits the AMR hierarchy itself.

One common way to organize volumes is through space-filling curves (e.g., Morton [14,15] or Hilbert codes [1,7]). In 3D rendering, the main incentive for that, as in the works cited here, is to build acceleration structures. Generally, space-filling curves cluster cells in 1D that are also nearby in 3D.

### 2.2 Dynamic Volume Lines

We extend the *dynamic volume lines* (DVL) method by Weissenböck *et al*. [13] to support large-scale AMR data. We provide a summary of their method in this section.

DVLs present volumes as 1D plots where the $x$-dimension maps to cell IDs and the y-dimension maps to intensity. The method is restricted to structured-regular volumes, i.e., all cells are cubes/voxels of the same size. One way to assign $x$-values is in a row- or column-major order. This approach loses spatial locality, as cells are only grouped if they are neighbors on the same row (or column); yet when they are adjacent in the vertical or depth direction, they are likely to be further apart in 1D due to the row (or column) sized stride.

Space-filling curves can provide better proximity-preserving mappings. Weissenböck *et al*. use Hilbert curves. Cells are represented by their centroids, which are quantized, e.g., to 20-bit per dimension, so that a 64-bit bitmask can represent them. The Hilbert codes represent the quantization grid cell that the centroids map to.

An obvious problem when plotting volumes in 1D is that there are several orders of magnitude more cells than pixels in the $x$-dimension. Using a linear mapping is wasteful because it can result in homogeneous or empty regions represented as straight lines that convey no useful information.

Weissenböck *et al*.'s objective is to compare ensembles of volumes. Interesting features are determined by comparing corresponding points of the ensemble. The authors propose scaling the cells along the $x$-axis using per-cell importance. However, their focus is on grayscale volumes and not on alpha transfer functions as we do. The difference between the maximum and minimum intensity of the whole ensemble gives the per-cell local variation of the ensemble:

$$V_h = \max_{\forall m \in M} \big(I(m,h)\big) - \min_{\forall m \in M} \big(I(m,h)\big), \tag{1}$$

where $h$ is the cell's Hilbert code, $M$ is the ensemble of volumes, and $I(m,h)$ is the intensity of ensemble member $m$ and the AMR cell that corresponds to $h$. From that, the authors obtain the local importance scale for the $x$-coordinate:

$$f(h) = \left( \frac{V_h}{\max\big(V_h\big)} \right)^P. \tag{2}$$

$P$ is a user-defined parameter to control the steepness of the resulting curve; a minimum importance is enforced (set to 0.025 by the authors).

Computing a prefix sum of floating point values over the per-cell importance and quantizing that on the 1D grid given by the width of the plot area provides $x$ positions to plot the cells as line segments. This nonlinear mapping compresses regions with low data variation, devoting more screen space to regions with high data variation.

The concept of exploring spatial or volume data in 1D also inspired other works. Franke *et al*. [4], e.g., use 1D plots to conserve neighborhood relations of geospatial regions. Zhou *et al*. [18] use the minimum spanning tree of a circuit graph over structured-regular volume ensembles to generate similar 1D plots as the volume lines our paper focuses on. In contrast to using Hilbert codes directly—and similar to DVLs with their importance-based nonlinear mapping—Zhou *et al*.'s approach is also data-driven.

## 3 Method

We extend Weissenböck *et al*.'s dynamic volume lines to support multi-field AMR data. In contrast to Weissenböck *et al*., we assume that the intensities $I(m,h)$ come from an RGB$\alpha$ transfer function. They do not focus on interactive parameter updates, such as transfer function, exponent $P$, and minimum importance; they mention that Hilbert code computation takes several seconds for $64^3$ cell data sets. While Weissenböck *et al*. focused on volumes a couple of Megabytes in size, we target Gigabyte-sized data.

### 3.1 Extension to AMR

To compute DVLs for AMR data, we need to extend the Hilbert code and $x$-coordinate generation to support non-uniformly sized cells. First, we note that Weissenböck *et al*. assume that cells have uniform size; even if the cells were non-uniform, the Hilbert codes do not reflect this because they only provide the cells' order and not their spacing. To account for coarser cells to also span a wider region of space as in a 3D rendering, we need to consider each cell's size when computing the nonlinear $x$-axis scale. For that, we extend Eq. (2) to include the AMR cell width as follows:

$$f(h) = \left( \frac{V_h}{\max\big(V_h\big)} 2^{L_h} \right)^P, \tag{3}$$

where $L_h \in \mathbb{N}_0$ is the AMR level of the cell corresponding to Hilbert code $h$, and 0 is the finest level. Another sensible choice would be to not scale the cells by their (uniform) width but by their volume.

We deliberately apply the cell-size dependent scale when computing the importance, not the $x$-positions themselves, because the order of operations is not commutative. We compute the floating point prefix sum over the importance values:

$$F(h) = \sum_{i=0}^{h} f(i), \tag{4}$$

to obtain $x$-coordinates $xf_1 = \frac{F(h-1)}{F(\max(h))} \times W$, $xf_2 = \frac{F(h)}{F(\max(h))} \times W$ for plots of size $W$ pixels.

### 3.2 Projecting Cells to 1D

We are now able to compute pairs $(xf_1, xf_2)$ of $x$-coordinates per AMR cell; we note that these $x$-coordinates have sub-pixel accuracy, and although we apply the nonlinear mapping using cumulative importance, in general, a multitude of $x$-coordinates will project to single pixels. We now discuss how to map these coordinate pairs to

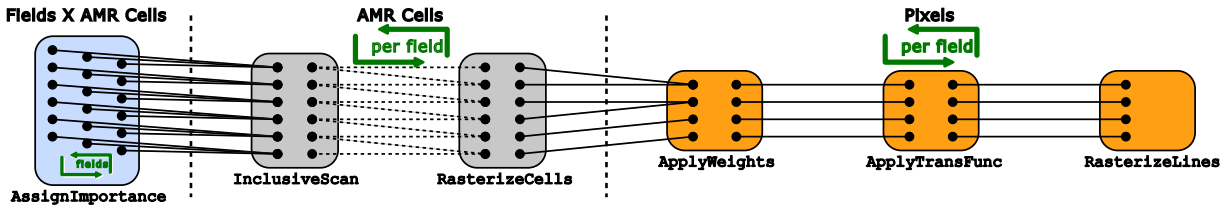


Figure 2: CUDA kernels executed on volume line updates. Blue indicates kernels processing each AMR cell and scalar field. Gray indicates kernel execution per AMR cell (these get executed *per* field). Orange boxes indicate kernels executed per (horizontal) raster point (also per field). As a rule of thumb, the number of work items generally decreases from left to right. Transitioning from "Fields $\times$ AMR Cells" to "AMR Cells" is realized using loops but requires no synchronization. Transitioning from "AMR Cells" to "Pixels" uses atomic operations on CUDA global memory.

obtain $x$-coordinates per horizontal pixel $x \in W$ and how to obtain "$y$-values" for these. Given such pairs $(x,y)$, we can draw line strips with control points per pixel in the $x$ dimension.

For that, we create a set of $W$ bins—one for each pixel in the plot's $x$-dimension—whose values we initialize to 0. We then project the pairs $(xf_1, xf_2)$ to integer coordinates in the range $[0, W-1]$, iterate over these, and increase the overlapping bins by the value of the corresponding AMR cell. We also maintain a per-bin counter that we increment whenever we increase the bin value. After all the pairs $(xf_1, xf_2)$ are processed, we iterate over the bins and divide each by its bin counter, obtaining the average intensity value of all the AMR cells that project to the bin.

This procedure borrows from the basis function method by Wald *et al*. [11], only that we use a box-shaped basis function instead of the tent-shaped basis used by Wald. Pairs $(xf_1, xf_2)$ that span multiple bins (hence multiple pixels in the $x$-dimension) will noticeably turn the plot into a step function. As for our data, many line segments will map to single bins only; we do not consider this to be an issue, and we here choose simplicity over generality.

### 3.3 Interactive Transfer Function Updates

We extend Weissenböck *et al*.'s method to support interactive RGB$\alpha$ transfer functions that apply to 1D and 3D rendering alike. The requirement that transfer function updates be interactive implies that (re)generating volume lines must also be interactive.

Transfer functions are applied on two occasions: once when computing the importance (cf. Section 3.1), which requires intensities from the transfer function, and once per bin, after dividing the bin values by their basis weights. We normalize the input (field) intensity and compute RGB$\alpha$ values to apply the transfer function. The alpha value determines the height of the bins and y-values of the polyline at these positions. The RGB value (or, alternatively, a uniform color from a global map) is used to colorize the polylines.

### 3.4 Computing the Maximum Local Ensemble Variation

The term $\max\left(V_h\right)$ from Eq. (2), the maximum of the local ensemble variations for each Hilbert code $h$, needs to be recomputed whenever an ensemble member's transfer function changes. This can be implemented on GPUs using parallel `reduce` over all cells or a kernel atomically updating a single value in GPU main memory.

To avoid this costly operation, we compute this value using the transfer functions and field data ranges, resulting in a more conservative, yet in practice very close approximation to $\max\left(V_h\right)$.

We compute ranges $[i_m, j_m]$ where $i_m, j_m \in [0, N-1]$ and $i_m \leq j_m$ for each ensemble member $m \in M$ and transfer function size $N$. These allow us to iterate only over the transfer function values present in the data. That way, we compute the global range $[i, j]$:

$$i = \min_{\forall m \in M}(i_m), j = \max_{\forall m \in M}(j_m), \tag{5}$$

and from that

$$V_a = \max_{\forall m \in M}\left(\mathrm{A}(m,a)\right) - \min_{\forall m \in M}\left(\mathrm{A}(m,a)\right) \; \forall a \in [i,j] \tag{6}$$

as an approximation to $V_h$. In Eq. (6), the term A denotes a lookup to the transfer function to retrieve the alpha value.

### 3.5 Brushing and Linking

We connect the DVL and 3D views using *brushing and linking*. When selecting regions of interest (ROIs) in the 1D plot, the corresponding cells in the 3D view are highlighted (cf. Fig. 3). We use the ROIs' first and last Hilbert codes as selection ranges for that. In the 3D shader, the ROIs also manifest as Hilbert codes and not as lists of cells in world space. This compact representation comes at the expense of transforming the center of the cell that we are sampling to Hilbert space to test it against the ROIs.

In the case of structured-regular volumes, finding the cell bounds is simple; since the cells have the same size, the corresponding Hilbert code implicitly allows us to derive their (uniform) size. For AMR data, the cell centroid's Hilbert code is not sufficient; instead, when we check if a sample falls inside an ROI, we must explicitly locate cells to know the ROI's exact size.

## 4 GPU Implementation with CUDA

We implement what we call *interactive volume lines* (IVLs)—the GPU-accelerated version of dynamic volume lines—using NVIDIA CUDA. The CUDA kernels involved are shown in Fig. 2, corresponding to the algorithm phases from Section 3. We now discuss how to assemble these building blocks and identify their bottlenecks. The Hilbert codes and the order of the 1D AMR cells never change once they are established. What does change in user interaction is the spacing between cells that we need to recompute interactively.

We compute the term $V_a$ from Eq. (6) on the CPU since the transfer function color map and other parameters are passed to our application through the host in any case and because the amount of computation needed does not necessitate running this operation in parallel. With $V_a$ computed, execution transitions to the GPU.

A kernel with one thread per AMR cell (`AssignImportance` in Fig. 2) computes the importance from Section 3.1. The kernel loops over each field, assigning each cell its transfer function and cell-size-dependent importance from Eq. (3). This requires exclusive read and write accesses (EREW) only. For Eq. (4), we use the `InclusiveSum` algorithm from the CUB library (`ExclusiveScan` in Fig. 2). GPU prefix sums can be realized with EREW accesses [6].

We run another kernel with one thread per cell (`RasterizeCells` in Fig. 2) to determine the subpixel $x$-coordinates $xf_1$,$xf_2$ from the prefix sum array and update the bins and weights (cf. Section 3.2). The number of bins is much smaller than the number of AMR cells for typical data sizes. We use CUDA atomic operations for the projection; hence, this kernel is not EREW. Due to the input size and the atomics, this is the most costly kernel of the algorithm.

The `ApplyWeights` and `ApplyTransFunc` kernels (cf. Fig. 2) are EREW. They divide the bin values by their weight (cf. Section 3.2) and apply the transfer function to obtain $y$-coordinates. Both kernels use $W$ (number of bins) threads. We finally run a kernel (`RasterizeLines` in Fig. 2) rasterizing the polylines using CUDA surfaces. We implemented modes to draw the IVLs as polylines (Fig. 1b) or as bar charts (Fig. 1c).

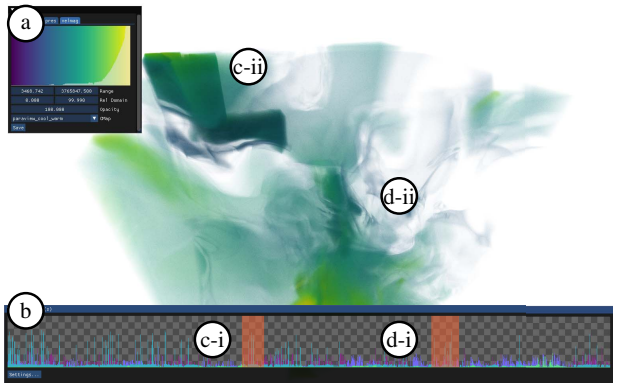


Figure 3: Our prototype, used with brushing and linking. Interactive transfer function updates (a) affect 3D rendering and the IVL plots shown in (b). Here we selected the two regions (c-i) and (d-i) in the IVL plot that are connected to the 3D view through brushing and linking and are highlighted using dimming ((c-ii) and (d-ii)).

Table 1: Execution times for the CUDA kernels from Fig. 2 and the data set from Fig. 1.

| Kernel | Execution Time (ms) |
|---|---|
| `AssignImportance` | 3 |
| `InclusiveScan` | 0.4 |
| `RasterizeCells` | 55 |
| `ApplyWeights` | $< 0.01$ |
| `ApplyTransFunc` | $< 0.01$ |
| `RasterizeLines` | 0.2 |

## 5 Prototypical User Interface

To implement a prototype, we started with the open-source code of Zellmann *et al.* [17] and added multi-field support and a 1D user interface with Dear ImGUI [2]. We show the user interface demonstrating brushing and linking in Fig. 3. We note that the application is at an early stage and does not implement all the features proposed by Weissenböck *et al.* [13], such as mouse-over handling or support for the histogram heatmap and functional boxplots. Fundamentally, the computations required are of the same order as those required for the IVLs, so implementing them remains an engineering exercise.

## 6 Evaluation

We evaluate our method on an Intel Xeon system with 64 GB RAM and an NVIDIA A6000 GPU. We use the `dens`, `temp`, `pres`, and `velmag` fields of the Molecular Cloud data set by Seifried *et al.* [10], which was simulated with FLASH [3]. Technically, the data is multi-field and not an ensemble; the fields are correlated. The data set spans four AMR levels with a total of 35.8 M cells. If not noted otherwise, we set $P = 1$ (cf. Eq. (3)) and the minimum importance to 0.025. GPU memory for the whole data set—including auxiliary data (our application uses OptiX to accelerate 3D rendering)—is reported by `nvidia-smi` to amount to 3.6 GB.

We present kernel execution times in ms. in Table 1. The algorithm is bottlenecked by the `RasterizeCells` kernel (projection from cells to bins using atomics), executed once per field.

We test if the performance of the `RasterizeCells` kernel depends on the input parameters. While keeping the other parameters fixed, we vary $P$ in $0.0 - 5.0$ (Fig. 4, left), and the minimum importance in $0.0 - 0.25$ (Fig. 4, right). We observe that there *is* a measurable, yet very subtle (1-2%) trend that parameters that favor higher IVL compression lead to slightly faster execution times.

We count the `atomicAdd`'s to determine their costs, and report the bin minima and maxima as well as quartiles for the same test from before (cf. Fig. 5). We observe that when $P$ and the minimum importance increase, the number of `atomicAdd`'s per bin becomes more uniform. We never encountered single threads that run extraordinarily long. In fact, during development, we tested if parallelizing the kernel using one `cudaStream` per field lowered total execution time, but found that it fully occupies the GPU at all times.

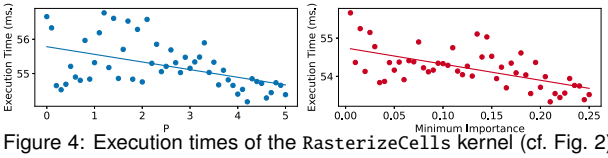


Figure 4: Execution times of the `RasterizeCells` kernel (cf. Fig. 2) as a function of parameter $P$ (left) and of minimum importance (right).

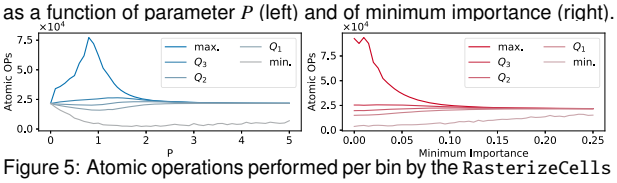


Figure 5: Atomic operations performed per bin by the `RasterizeCells` kernel (cf. Fig. 2) as a function of parameter $P$ (left) and of minimum importance (right). We show the minimum and maximum count as well as the quartiles across bins.

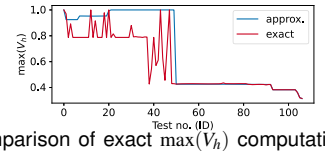


Figure 6: Comparison of exact $\max(V_h)$ computation (red) vs. our approximation from Section 3.4 (blue).

We finally compute the difference between the exact $\max(V_h)$ used by Weissenböck *et al.*'s [13] vs. our approximation from Section 3.4, using a sweep of 107 randomly chosen different transfer function configurations. We report the results in Fig. 6.

## 7 Conclusions and Future Work

We presented *interactive* volume lines, a high-performance variant of Weissenböck *et al.*'s [13] dynamic volume lines. For non-trivial volume data (e.g., unstructured or AMR), visual analytics performed on the whole set of cells becomes a complex data handling task. We presented a carefully crafted GPU implementation of this algorithm.

For large-scale volumes, as demonstrated in this paper, it does matter if an operation is performed on the whole set of cells, on bins of a 1D grid in screenspace, or on the RGB$\alpha$ transfer function array, and which type of write accesses are performed. The GPU control flow we eventually developed resulted in a balance between faithfully recreating the algorithm and avoiding severe implementation bottlenecks.

Our paper points to future work. One conceivable optimization is to make the kernel that the algorithm is bottlenecked on become hierarchical over the input cells to perform the projection in more local memory regions. This is one of many interesting similarities between optimizations used for 3D volume rendering (e.g., acceleration structures like the one by Wald *et al.* [12]) and optimizations that apply to IVLs. In fact, we found that on the engineering side, similar abstractions apply in 1D and 3D alike (e.g., basis functions used for interpolation, 1D bins that resemble pixels, etc.).

One less obvious future work lies in the algorithm's GPU memory consumption: 1D and 3D rendering share neither the data itself nor other auxiliary data structures that accelerate rendering. Instead, the whole input data is replicated in memory. It would be interesting to explore if the two rendering modes could share some or even all the data to overcome this limitation.

Another open question (that this paper does not seek to answer) is how useful volume lines are for large volumes; our intuition is that they work best for smaller data, to reduce noise and outliers in the 1D plots. A formal evaluation is out of scope here. We note though that an obvious extension would be a zoom interaction that, when selecting a ROI through brushing and linking, also zooms in on the 1D plot and devotes more screen space to the selected AMR cells.

## ACKNOWLEDGMENTS

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—grant no. 456842964. The TAC Molecular Cloud is courtesy of Daniel Seifried. We are grateful to NVIDIA, who kindly provided us with the hardware we used for the evaluation.

## REFERENCES

[1] A. Aman, S. Demirci, and U. Güdükbay. Compact tetrahedralization-based acceleration structures for ray tracing. *Journal of Visualization*, 25(5):1103–1115, 2022.

[2] O. Cornut. Dear ImGui: Bloat-free graphical user interface for C++ with minimal dependencies. `https://github.com/ocornut/imgui`, 2014. Accessed: 2023-04-29.

[3] A. Dubey, R. Fisher, C. Graziani, I. Jordan, G. C., D. Q. Lamb, L. B. Reid, P. Rich, D. Sheeler, D. Townsley, and K. Weide. Challenges of extreme computing using the FLASH code. In N. V. Pogorelov, E. Audit, and G. P. Zank, eds., *Numerical Modeling of Space Plasma Flows*, vol. 385 of *Astronomical Society of the Pacific Conference Series*, p. 145, Apr. 2008.

[4] M. Franke, H. Martin, S. Koch, and K. Kurzhals. Visual analysis of spatio-temporal phenomena with 1D projections. *Computer Graphics Forum*, 21(3):335–357, 2021. doi: 10.1111/cgf.14311

[5] 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. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2285–2294, 2012.

[6] M. Harris, S. Sengupta, and J. D. Owens. Parallel prefix sum (scan) with CUDA. In H. Nguyen, ed., *GPU Gems 3*, chap. 39, pp. 851–876. Addison Wesley, August 2007.

[7] N. Morrical, A. Sahistan, U. Güdükbay, I. Wald, and V. Pascucci. Quick Clusters: A GPU-parallel partitioning for efficient path tracing of unstructured volumetric grids. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):537–547, 2023. doi: 10.1109/TVCG.2022.3209418

[8] P. Muigg, M. Hadwiger, H. Doleisch, and E. Groller. Interactive volume visualization of general polyhedral grids. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2115–2124, 2011.

[9] A. Sahistan, S. Demirci, N. Morrical, S. Zellmann, A. Aman, I. Wald, and U. Güdükbay. Ray-traced shell traversal of tetrahedral meshes for direct volume visualization. In *Proceedings of IEEE Visualization Conference*, VIS '21, pp. 91–95, 2021.

[10] D. Seifried, S. Walch, P. Girichidis, T. Naab, R. Wünsch, R. S. Klessen, S. C. O. Glover, T. Peters, and P. Clark. SILCC-Zoom: the dynamic and chemical evolution of molecular clouds. *Monthly Notices of the Royal Astronomical Society*, 472(4):4797–4818, Dec 2017. doi: 10.1093/mnras/stx2343

[11] I. Wald, C. Brownlee, W. Usher, and A. Knoll. CPU volume rendering of adaptive mesh refinement data. In *Proceedings of the SIGGRAPH Asia 2017 Symposium on Visualization*, 2017. Article no. 9. doi: 10.1145/3139295.3139305

[12] I. Wald, S. Zellmann, W. Usher, N. Morrical, U. Lang, and V. Pascucci. Ray tracing structured AMR data using ExaBricks. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):625–634, 2021. doi: 10.1109/TVCG.2020.3030470

[13] J. Weissenböck, B. Fröhler, E. Gröller, J. Kastner, and C. Heinzl. Dynamic Volume Lines: Visual comparison of 3D volumes through space-filling curves. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1040–1049, 2019. doi: 10.1109/TVCG.2018.2864510

[14] S. Zellmann, M. Hellmann, and U. Lang. A linear time BVH construction algorithm for sparse volumes. In *2019 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 222–226, 2019. doi: 10.1109/PacificVis.2019.00033

[15] S. Zellmann, J. P. Schulze, and U. Lang. Binned k-d tree construction for sparse volume data on multi-core and GPU systems. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):1904–1915, 2021. doi: 10.1109/TVCG.2019.2938957

[16] S. Zellmann, I. Wald, A. Sahistan, M. Hellmann, and W. Usher. Design and evaluation of a GPU streaming framework for visualizing time-varying AMR data. In R. Bujack, J. Tierny, and F. Sadlo, eds., *Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization*, PGV '22. The Eurographics Association, 2022. doi: 10.2312/pgv.20221066

[17] S. Zellmann, Q. Wu, K.-L. Ma, and I. Wald. Memory-Efficient GPU Volume Path Tracing of AMR Data Using the Dual Mesh. *Computer Graphics Forum*, 42(3), 2023. doi: 10.1111/cgf.14811

[18] L. Zhou, C. R. Johnson, and D. Weiskopf. Data-driven space-filling curves. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1591–1600, 2021. doi: 10.1109/TVCG.2020.3030473