Efficient Space Skipping and Adaptive Sampling of Unstructured Volumes Using Hardware Accelerated Ray Tracing

Nate Morrical[1] Will Usher[1] Ingo Wald[2] Valerio Pascucci[1]
[1]SCI Institute, University of Utah [2]NVIDIA

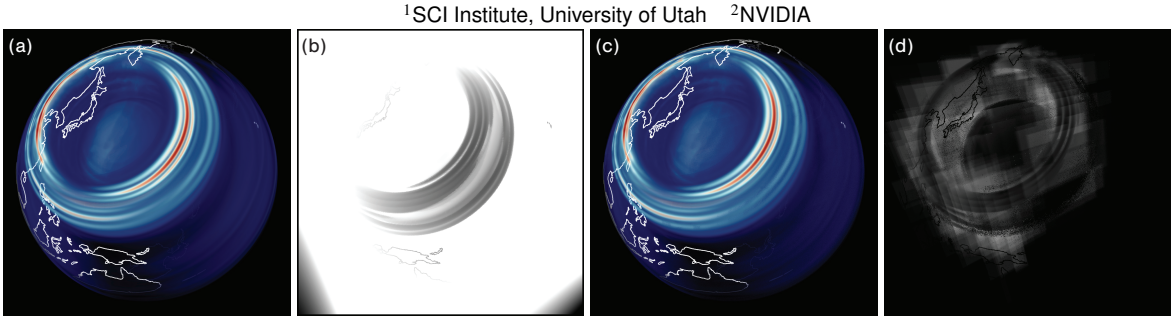


Figure 1: Performance improvement of our method on the 278 million tetrahedra Japan Earthquake data set. (a) A reference volume ray marcher without our method, at 0.9 FPS (1024^2 pixels) on an NVIDIA RTX 8000 GPU. (b) A heat map of relative cost per-pixel in (a). (c) and (d), the same, but now with our space skipping and adaptive sampling method, running at 7 FPS (7× faster).

ABSTRACT

Sample based ray marching is an effective method for direct volume rendering of unstructured meshes. However, sampling such meshes remains expensive, and strategies to reduce the number of samples taken have received relatively little attention. In this paper, we introduce a method for rendering unstructured meshes using a combination of a coarse spatial acceleration structure and hardware-accelerated ray tracing. Our approach enables efficient empty space skipping and adaptive sampling of unstructured meshes, and outperforms a reference ray marcher by up to 7×.

Keywords: Volume rendering, space skipping, adaptive sampling

1 INTRODUCTION

Direct volume rendering (DVR) is widely used in the scientific visualization community, enabling scientists to interactively explore their data and form hypotheses. A standard approach for DVR is raycasting, where rays are cast through the volume for each pixel and the color and opacity of the volume is sampled and integrated along each ray to compute an image of the volume. Interactive ray casting techniques have been demonstrated for both structured [15, 23] and unstructured [21, 22, 25, 32, 34] volumes, and map well to the parallel hardware available on modern CPUs [23, 25, 31] and GPUs [15, 21, 22, 32, 34].

However, when the volume becomes expensive to sample the cost per-ray increases, limiting interactivity. For unstructured data, the cost of these samples can be reduced, as demonstrated by Wald et al. [32], by leveraging the ray tracing cores available on NVIDIA's Turing GPUs. When used in a naive volume raycaster, their approach improved frame rates by 1.5× to 3.8×. To further improve performance, the number of samples taken per-ray must be reduced. Numerous methods have been proposed for regular grid volumes, which roughly fall into two categories: empty-space skipping, which avoids sampling fully transparent regions; and adaptive sampling, which takes fewer samples in regions containing less interesting data values. Prior work has employed a range of acceleration structures to enable these optimizations, e.g., macrocells [15, 23], octrees [1, 11, 16, 17, 26, 35], KD-trees [29, 30] and BVHs [14].

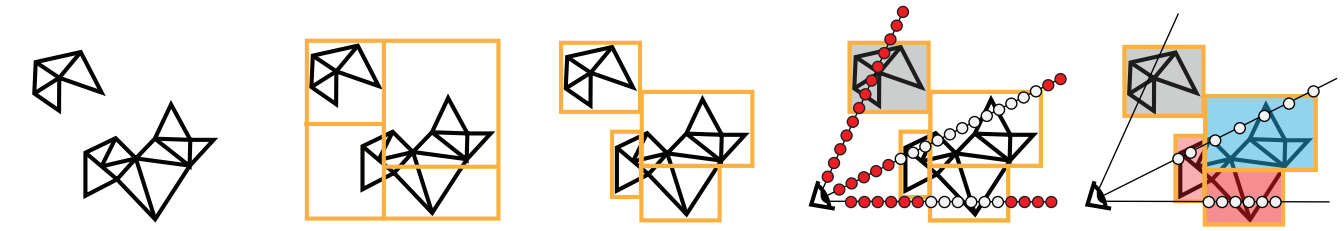
When considering an additional acceleration structure for DVR, the performance overhead introduced by building and traversing the structure is of key concern. If the overhead incurred by the structure is too high, it can overshadow the performance gained from the reduced number of samples taken. To reduce this overhead, Hadwiger et al. [13] proposed SparseLeap, which leverages triangle rasterization hardware to compute per-pixel lists of active ray segments by rendering occupancy geometry. These segments act as the space skipping acceleration structure in a subsequent render pass. Ganter et al. [8] recently extended SparseLeap to leverage OptiX [24] and NVIDIA's ray tracing cores to improve acceleration structure build time and use hardware accelerated BVH traversal to reduce overhead. However, both methods must rebuild the structure on transfer function changes, do not consider adaptive sampling, and rely on either an octree or a summed area table to build the occupancy geometry, which can result in poor adaptivity to an unstructured mesh.

In the context of unstructured meshes, relatively little prior work has investigated object space adaptive sampling or empty space skipping. The bulk of methods for rendering such volumes focus on rasterization methods [3, 5, 19, 20, 27], requiring either dynamic level-of-detail strategies [4, 6] or a volume simplification pre-processing step [9, 18, 28] to achieve adaptive sampling. Standard approaches for ray tracing such data [2, 10, 12, 22] step from cell-to-cell along the ray, providing an accurate image at significant cost. Nelson et al. [22] skip individual empty mesh elements, but do not support skipping larger regions at once. Prior work has also proposed rasterizing proxy meshes to perform ray tracing on the GPU [21, 34], but encounter similar adaptive sampling challenges with regard to which proxy geometry to dispatch. Sample based ray casting methods [25] have shown performance improvements over both rasterization and cell iteration methods, though have not investigated empty space skipping or adaptive sampling.

In this work, we propose new strategies for empty space skipping and adaptive sampling for sample-based raycasting of unstructured meshes. In contrast to prior work, our empty space skipping structure allows skipping larger regions of space and adapts well to the underlying mesh. Moreover, we formulate our space skipping structure in a manner suitable for hardware acceleration, without requiring a rasterizer. Finally, we propose an intuitive adaptive sampling approach for occupancy geometry methods without requiring additional preprocessing. Our contributions are:

- An extension of "occupancy geometry" to unstructured data;
- A lightweight empty space skipping method for unstructured data, which leverages GPU ray tracing hardware; and
- An adaptive sampling approach which provides a bound on error using three intuitive parameters.

Author's copy - to appear in IEEE VIS 2019 conference proceedings and IEEE Xplore.



(a) Input unstructured mesh (b) Build KD-tree (c) Shrink partitions (d) Empty space skipping (e) Adaptive sampling

Figure 2: An illustration of our method: (a) Given an unstructured mesh; (b) we build a coarse spatial subdivision over the mesh elements to partition them into a set of convex, disjoint regions. (c) These partitions are then shrunk to tightly fit the contained elements. For each region we compute the min/max of the scalar field and the transfer function maximum opacity and color variance. (d) We use hardware accelerated ray tracing to iterate rays through the "active" partitions, skipping transparent ones and unoccupied space. (e) Within each partition we use the local variance to adapt the sampling rate to the underlying data variation, thereby taking fewer samples in more uniform regions of the data.

2 Method

Unstructured meshes pose unique challenges to implementing an effective empty space skipping and adaptive sampling strategy. Such meshes are often refined by the simulation in areas of interest, and the method must adapt to this non-uniformity, where the size and distribution of the elements is not known a priori. Furthermore, as the mesh is not guaranteed to be convex, the method must account for three categories of coarser regions: unoccupied regions containing no elements, regions containing entirely transparent elements, and regions where the field does not vary significantly across elements.

In our method, we first partition the volume into a set of convex, disjoint regions (Section 2.1), which are shrunk to tightly fit the contained mesh elements (Section 2.2). We then compute metadata about the elements contained in these partitions, which we use to guide both the empty-space skipping (Section 2.3) and adaptive sampling (Section 2.4). Finally, we use the Turing GPU's new ray tracing hardware to accelerate traversal of these partitions for empty space skipping, and our adaptive sampling method to reduce the number of samples taken in each partition. An overview of our method is shown in Figure 2.

2.1 Partition Generation

To partition the mesh elements, we use the leaves of a spatial KD-tree (Figure 2b), which are convex, disjoint, and adaptive. The disjoint, non-overlapping property ensures that a ray will exit one partition before entering the next, and the convexity property ensures that the ray will enter each partition only once. As the elements in the mesh are unlikely to be uniform in size or distribution, the adaptivity in partition size provided by the KD-tree is desirable over a more fixed structure (e.g., grids [8], octrees [13]) to ensure a roughly even distribution of rendering cost for the generated partitions. We note that it is possible for a tetrahedron (or other mesh element) to appear in more than one leaf node, and thus partition. In this case, rays entering a given partition will only sample the portion of the element contained inside the partition's bounds.

For each partition we compute and store the scalar field range of the contained elements. For those elements which are only partly contained in the partition we include the value range of the entire element for simplicity. When the transfer function is changed we apply it across the field range of the partition to determine the maximum opacity and color variance of the partition. The complexity of this computation is linear in the number of values in the transfer function, and is independent of the number of elements contained in the partition. As there are likely far fewer values in the transfer function than elements in the mesh, this provides better responsiveness to user changes to the transfer function. The variance computation is parallelized over the partitions, allowing for faster updates. The per-partition variance values are normalized relative to the minimum and maximum variances over all partitions, to compute consistent per-partition sampling rates.

2.2 Partition Refinement

Although the KD-tree leaves provide the desired convex and disjoint partitioning of the volume, the bounds of a leaf do not necessarily tightly bound the contained elements (see Figure 2b). As such, the initially computed partitions may contain large regions of unoccupied space, especially in the case of non-convex, hollow, or non-grid aligned meshes. To provide tighter bounds, we shrink the bounding box of each partition to the intersection of the original bounds and the bounds of the contained elements (Figure 2c).

2.3 Empty Space Skipping

Given the bounds of the partitions, we can leverage hardware accelerated ray tracing to accelerate intersection tests against the partitions to find ray entry and exit points. The RT cores available in Turing GPUs support both hardware accelerated BVH traversal and ray-triangle intersection tests. To fully utilize the available hardware, we first tessellate the partition bounding boxes, then construct an OptiX [24] BVH over the generated triangles. This BVH need only be rebuilt when the underlying partition geometry changes, and is not tied to the transfer function.

To find the entry and exit points of the ray in a partition we use OptiX to trace rays against the partition geometry; first with back-face culling enabled to find the entry point, then from the entry point with front-face culling enabled to find exit point. The t range along the ray between t_{enter} and t_{exit} is thus the range to integrate the volume over to sample the partition. If the ray intersects a completely transparent partition it is skipped and we advance the ray to find the next partition. If no partition is found, the ray is terminated and the computed color and opacity written to the framebuffer.

To advance to the next partition we set the ray's t_{min} to $t_{\text{exit}} - \varepsilon$. We apply a small offset back by ε to allow for intersection with potentially coplanar partition boundary faces. From this new start point we then find the ray's entry and exit points with the next partition as before.

2.4 Adaptive Sampling

To adaptively sample each partition we use the transfer function variance for the partition computed in Section 2.1 to select the step size for the ray marching process. In partitions with relatively similar colors the variance is low, and a correspondingly low number of samples can be taken. Regions with higher color variance require correspondingly more samples to preserve accuracy. The step size is computed using an intuitive equation which allows users to place an upper bound on the tolerable maximum step size, and thus error; and control how quickly the algorithm transitions from high to low quality sampling based on the partition's variance. Given a minimum step size s_1 to use for the highest quality sampling and the maximum step size s_2 to use for the lowest quality sampling, we compute the step size for a partition with normalized variance σ^2 using $s = \max\left(s_1 + (s_2 - s_1)|\min(\sigma, 1) - 1|^p, s_1\right)$.

The final user controllable parameter is p, referred to as the *adaptive power*, which allows the user to tune how quickly the algorithm

will transition to lower quality sampling in medium variance partitions. We restrict that $p \geq 1$, as this lower bound corresponds to a simple linear interpolation between s_1 and s_2.

With this equation the user can easily tune the sampling quality to produce an acceptable image at some desired frame rate. If the user wants a lower quality image at a higher frame rate they can increase s_1 and s_2, or for a more expensive but higher-quality image, decrease both values. If desired, the adaptive sampling can be disabled entirely by setting $s_1 = s_2$. If the user finds too few samples are taken in partitions with medium variance they can increase p to bias s towards s_1 in these partitions. Similarly, to improve frame rate at the cost of error in medium variance partitions p can be decreased, to bias s towards s_2.

Given the sampling step size for a partition we integrate the ray front to back through the partition, using the `rtx-shared-faces` point query kernel described by Wald et al. [32] to sample at points along the ray. To ensure correct opacity when compositing partitions integrated at different step sizes we use an opacity correction term [7]. Given the current sample's opacity α we compute the corrected opacity as $\tilde{\alpha} = 1 - (1 - \alpha)^{s/s_1}$. Finally, we perform early ray termination if the ray becomes opaque.

3 Evaluation

We evaluate our approach using four tetrahedral mesh volumes (Figure 3) covering a range of data set sizes, on an NVIDIA RTX 8000 GPU, primarily due to its large memory capacity. Our renderer is implemented using OptiX 6 and CUDA 10. For the Jets, Agulhas Current and Japan Earthquake datasets we set $p = 2$, on the Deep Water Asteroid Impact we set $p = 6$. We first evaluate the performance gains provided by our empty space skipping method (Section 3.1), after which we combine it with our adaptive sampling method and evaluate the two in combination (Section 3.2). Finally, we measure the overhead incurred by the two methods in Section 3.3.

3.1 Space Skipping Improvements

Empty space skipping is able to reduce the samples taken per-ray in two ways: by skipping regions outside the volume, and by skipping 100% transparent partitions. These regions can be skipped, since they do not contribute to the final image (Figure 5b).

With regard to the former, we observe a negligible performance improvement when only skipping unoccupied space compared to naively taking samples potentially outside the volume. The volumes we conduct our evaluation on are relatively dense, providing little unoccupied space to skip in the first place. Moreover, it is likely that the hardware accelerated BVH traversal performed when querying a point is able to quickly determine the point is outside the volume and terminate, incurring little cost per-sample.

The performance improvement provided in the latter case, skipping 100% transparent partitions, is highly dependent on the transfer function chosen by the user (Figure 6). When using a relatively "binary" transfer function, where background regions are made entirely transparent, a large number of partitions can be discarded, yielding a significant performance improvement. However, if these background regions are made slightly opaque it is no longer possible to discard them, and relatively few empty partitions can be skipped, thereby limiting the performance improvement which can be achieved.

3.2 Adaptive Sampling Improvements

Our adaptive sampling approach can provide significant performance improvements by reducing samples taken in low variance partitions (Figure 5c), with little sacrifice in image quality. When combined with our empty space skipping approach, adaptive sampling improves performance in semitransparent low-variance regions (see Figure 6). For high-quality rendering using both methods in combination we find significant performance improvements. On the Jets, Agulhas Current and Deep Water Asteroid Impact we achieve a roughly $3.5\times$ improvement in rendering performance; on the Japan Earthquake we achieve a $7.8\times$ improvement; in all cases SSIM ≥ 0.97 (Figure 3).

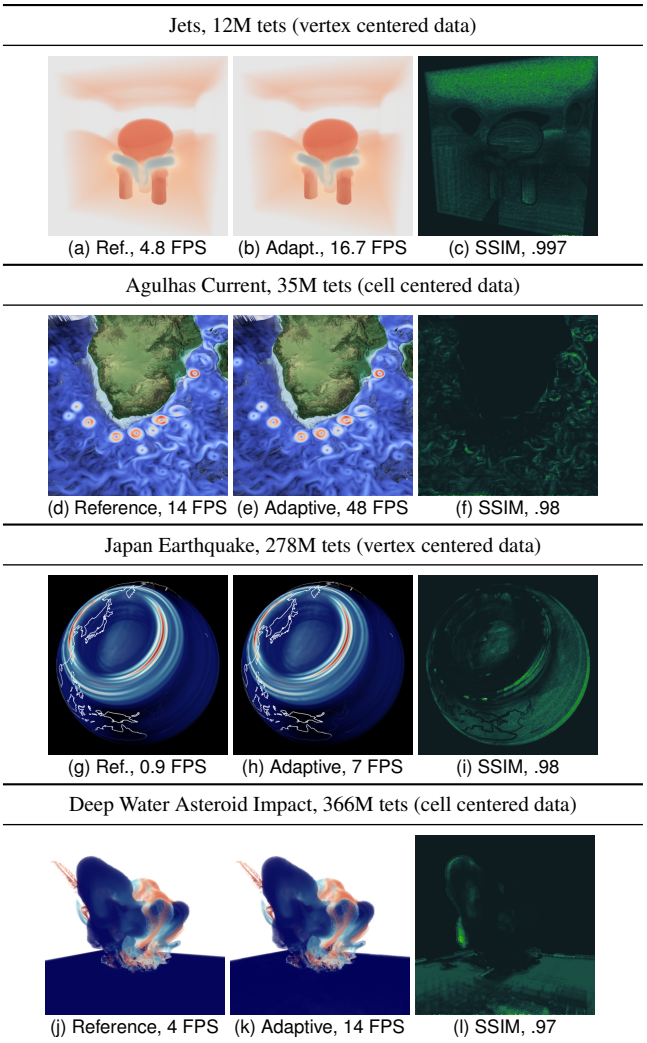


Figure 3: Quality and performance comparisons of our method against a reference volume ray marcher [32], using representative views and transfer functions. For comparable image quality our method performs roughly $3-7\times$ faster, achieving its greatest speedup in the most irregular data set (Japan Earthquake). More aggressive quality settings can yield higher speedups, at the cost of image quality. For larger images, please see the supplemental material.

In Figure 4 we evaluate the impact on frame rate, samples taken and image quality as the tolerable maximum step size s_2 is increased. As s_2 is increased, the adaptive sampling can take larger steps in low and medium variance regions, reducing samples taken and thereby improving performance, at the cost of image quality. At the extreme end we find that even when taking $1/3$ or fewer samples than the reference, a tolerable medium to high-quality image can still be provided. Moreover, significant reductions in samples taken, and thus increases in performance, can be achieved with little perceivable impact on image quality. For volumes with more expensive sampling, e.g., higher-order interpolants or non-hardware accelerated tet-mesh sampling, the performance improvement achieved by reducing the number of samples taken is likely to be even greater.

3.3 Acceleration Structure Overhead

In our evaluation we found the added work of intersecting rays with the partition boundaries is negligible. With relatively few generated

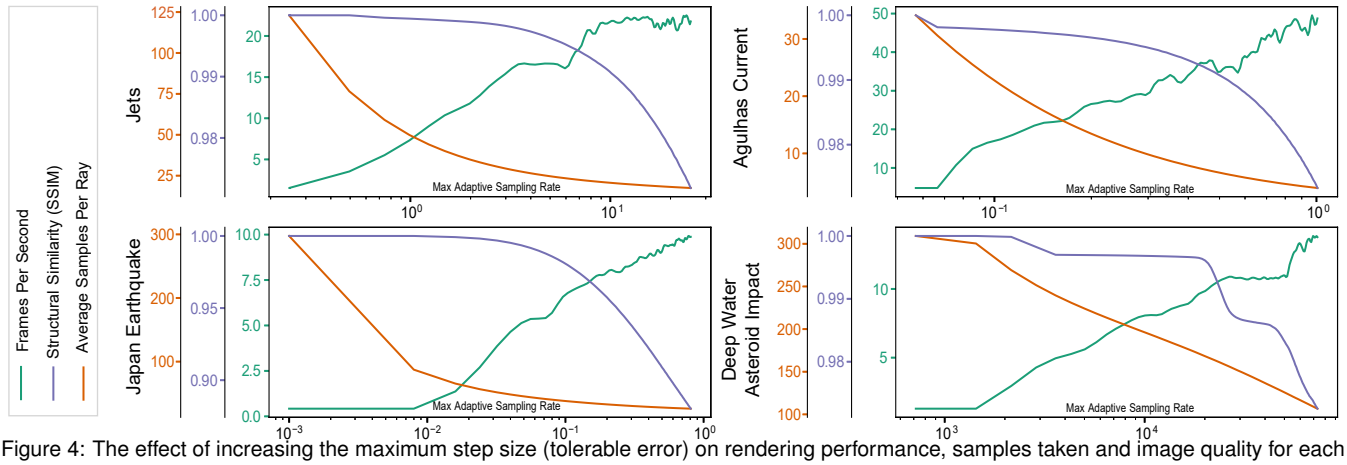
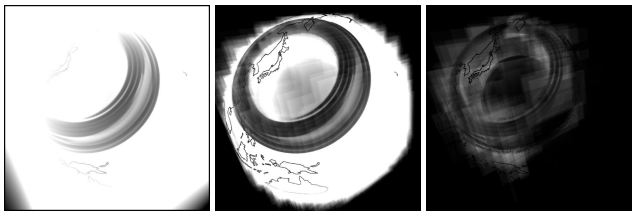
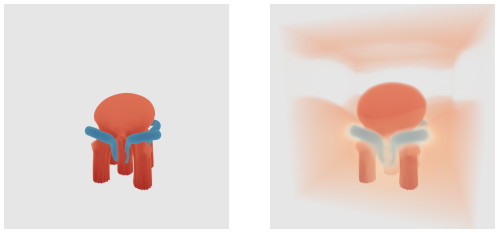


Figure 4: The effect of increasing the maximum step size (tolerable error) on rendering performance, samples taken and image quality for each data set. As expected, image quality decreases as the maximum step size is increased, though remains high-quality (SSIM $\geq$ 0.97) even when taking just a fraction of the original samples required. Using our approach users can tune the error incurred as desired to achieve improved frame rates for unstructured volume rendering.



(a) Reference (b) Space skipping only. (c) Plus adapt. sampling.

Figure 5: A heatmap of the samples taken per-pixel compared to the reference (a), when using (b) just space skipping, and (c) space skipping plus adaptive sampling. (a) The reference takes a large number of samples for most pixels, except a few where early ray termination occurs. (b) Space skipping avoids unoccupied and fully transparent partitions, though takes many samples in visible partitions. (c) Adaptive sampling reduces samples taken in visible low-variance regions, reducing samples while providing similar image quality.



Reference: 4 FPS
Space skipping only: 20 FPS
Adaptive only: 10 FPS
Together: 24 FPS

Reference: 4 FPS
Space skipping only: 5 FPS
Adaptive only: 12 FPS
Together: 13 FPS

Figure 6: Empty space skipping works best for "binary" transfer functions, where parts of the volume are 100% transparent (left); but on its own breaks down if regions are not 100% transparent (right). Adaptive sampling is able to reduce the sampling rate in these semitransparent low-variance regions, improving performance for such cases.

partitions, even for large data sets, and hardware accelerated ray tracing used to intersect these partitions, there is little cost incurred to traverse them. For example, on the Japan Earthquake, which contains the largest number of partitions (4725), we find tracing rays through these partitions takes just 3ms.

4 Limitations, Future Work, and Conclusion

We have presented new strategies for leveraging empty space skipping and adaptive sampling in the context of volume rendering unstructured meshes. Our method significantly reduces the number of samples required per-pixel, improving performance without sacrificing image quality. Our adaptive sampling method exposes an intuitive set of parameters to users, allowing them to easily control the trade off between performance and image quality. Furthermore, our approach is able to leverage the new ray tracing hardware available on recent GPUs and incurs little overhead as a result.

Although we have demonstrated significant performance improvements when using our method, it is not without limitations. First, as with other adaptive sampling approaches we find diminishing returns as the sampling rate becomes very low. Second, our adaptive sampling is only based on the transfer function, and would require additional metadata per partition to account for gradient shading or scattering. Third, our occupancy geometry must be rebuilt on mesh geometry changes, which would impact performance on both time series data as well as runtime level-of-detail strategies. On large datasets such as the Deep Water Asteroid Impact, we encountered numerical precision issues when traversing the partitions. These precision issues may be addressed by adapting our epsilon offset to account for the data set size, or using a custom higher-precision intersection test for the partition geometry. Finally, although the median split KD-tree provides a reasonable spatial partitioning, taking into account the underlying scalar field may provide better results. For example, on the Deep Water Asteroid Impact the water is split into multiple partitions; however, the field is uniform across the region and a single partition would suffice.

In future work, it would be valuable to explore a method for automatically selecting the adaptive sampling parameters based on some runtime refinement to provide a desired image quality, instead of requiring users to manually tune the sampling parameters. Although we have evaluated our method in a GPU raycaster to leverage hardware accelerated ray tracing, it could translate well onto the CPU using Embree [33]. While we have only evaluated our method on linearly interpolated tetrahedral meshes, a similar approach may work well for other unstructured volumes, adaptive mesh refinement (AMR) volumes, and higher order interpolants.

Acknowledgments

The Agulhas is courtesy Dr. Niklas Röber (DKRZ); the Japan Earthquake is courtesy of Carsten Burstedde, Omar Ghattas, James R. Martin, Georg Stadler, and Lucas C. Wilcox (ICES, UT Austin) and Paul Navrátil and Greg Abram (TACC); the Deep Water Asteroid Impact is courtesy of John Patchett and Galen Gisler of LANL. Hardware for development and testing was graciously provided by NVIDIA Corp. This work is supported in part by NSF: CGV Award: 1314896, NSF:IIP Award: 1602127, NSF:ACI Award: 1649923, DOE/SciDAC DESC0007446, CCMSC DE-NA0002375 and NSF:OAC Award: 1842042.

REFERENCES

[1] I. Boada, I. Navazo, and R. Scopigno. Multiresolution Volume Visualization with a Texture-Based Octree. *The Visual Computer*, 2001.

[2] P. Bunyk, A. Kaufman, and C. T. Silva. Simple, Fast, and Robust Ray Casting of Irregular Grids. In *Scientific Visualization Conference (Dagstuhl '97)*, 1997.

[3] S. Callahan, M. Ikits, J. Comba, and C. Silva. Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2005.

[4] S. P. Callahan. The k-buffer and its applications to volume rendering, 2005.

[5] S. P. Callahan. Adaptive visualization of dynamic unstructured meshes, 2008.

[6] R. Cook, N. Max, C. T. Silva, and P. L. Williams. Image-space visibility ordering for cell projection volume rendering of unstructured data. *IEEE Transactions on Visualization and Computer Graphics*, Nov 2004.

[7] K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-Time Volume Graphics*. 2006.

[8] D. Ganter and M. Manzke. An Analysis of Region Clustered BVH Volume Rendering on GPU. *Computer Graphics Forum*, 2019.

[9] M. Garland and Y. Zhou. Quadric-based simplification in any dimension. *ACM Trans. Graph.*, 2005.

[10] M. P. Garrity. Raytracing irregular volume data. *ACM SIGGRAPH Computer Graphics*, 1990.

[11] E. Gobbetti, F. Marton, and J. A. Iglesias Guitián. A Single-Pass GPU Ray Casting Framework for Interactive Out-of-Core Rendering of Massive Volumetric Datasets. *The Visual Computer*, 2008.

[12] G. Gu and D. Kim. Accurate and Memory-Efficient GPU Ray-Casting Algorithm for Volume Rendering Unstructured Grid Data. In J. Madeiras Pereira and R. G. Raidou, eds., *EuroVis 2019 - Posters*, 2019.

[13] M. Hadwiger, A. K. Al-Awami, J. Beyer, M. Agus, and H. Pfister. SparseLeap: Efficient empty space skipping for large-scale volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[14] A. Knoll, S. Thelen, I. Wald, C. D. Hansen, H. Hagen, and M. E. Papka. Full-resolution interactive CPU volume rendering with coherent BVH traversal. In *Visualization Symposium (PacificVis)*, 2011.

[15] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS '03)*, 2003.

[16] M. Labschütz, S. Bruckner, M. E. Gröller, M. Hadwiger, and P. Rautek. JiTTree: A Just-in-Time Compiled Sparse GPU Volume Data Structure. *IEEE Transactions on Visualization and Computer Graphics*, 2016.

[17] E. LaMar, B. Hamann, and K. I. Joy. Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In *VIS '99 Proceedings of the Conference on Visualization '99: Celebrating Ten Years*, 1999.

[18] J. Leven, J. Corso, J. Cohen, and S. Kumar. Interactive visualization of unstructured grids using hierarchical 3d textures. In *Symposium on Volume Visualization and Graphics, 2002. Proceedings. IEEE / ACM SIGGRAPH*, Oct 2002.

[19] A. Maximo, R. Marroquim, and R. Farias. Hardware-Assisted Projected Tetrahedra. *Computer Graphics Forum*, 2010.

[20] K. Moreland and E. Angel. A fast high accuracy volume renderer for unstructured data. In *2004 IEEE Symposium on Volume Visualization and Graphics*, 2004.

[21] P. Muigg, M. Hadwiger, H. Doleisch, and E. Gröller. Interactive Volume Visualization of General Polyhedral Grids. *IEEE Transactions on Visualization and Computer Graphics*, 2011.

[22] B. Nelson and R. M. Kirby. Ray-tracing polymorphic multidomain spectral/hp elements for isosurface rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2006.

[23] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, and C. Hansen. Interactive Ray Tracing for Volume Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 1999.

[24] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, and A. Robison. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 2010.

[25] B. Rathke, I. Wald, K. Chiu, and C. Brownlee. SIMD Parallel Ray Tracing of Homogeneous Polyhedral Grids. In *Eurographics Symposium on Parallel Graphics and Visualization*, 2015.

[26] F. Reichl, M. Treib, and R. Westermann. Visualization of Big SPH Simulations via Compressed Octree Grids. In *2013 IEEE International Conference on Big Data*, 2013.

[27] P. Shirley and A. Tuchman. A Polygonal Approximation to Direct Scalar Volume Rendering. In *Proceedings of the 1990 Workshop on Volume Visualization*, 1990.

[28] C. Silva, J. Comba, S. P. Callahan, and F. F. Bernardon. A survey of gpu-based volume rendering of unstructured grids. *RITA*, 12, 01 2005.

[29] K. R. Subramanian and D. S. Fussell. Applying space subdivision techniques to volume rendering. In *VIS '90 Proceedings of the 1st Conference on Visualization*, 1990.

[30] V. Vidal, X. Mei, and P. Decaudin. Simple Empty-Space Removal for Interactive Volume Rendering. *Journal of Graphics Tools*, 2008.

[31] I. Wald, G. P. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Günther, and P. Navrátil. OSPRay – A CPU Ray Tracing Framework for Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2017.

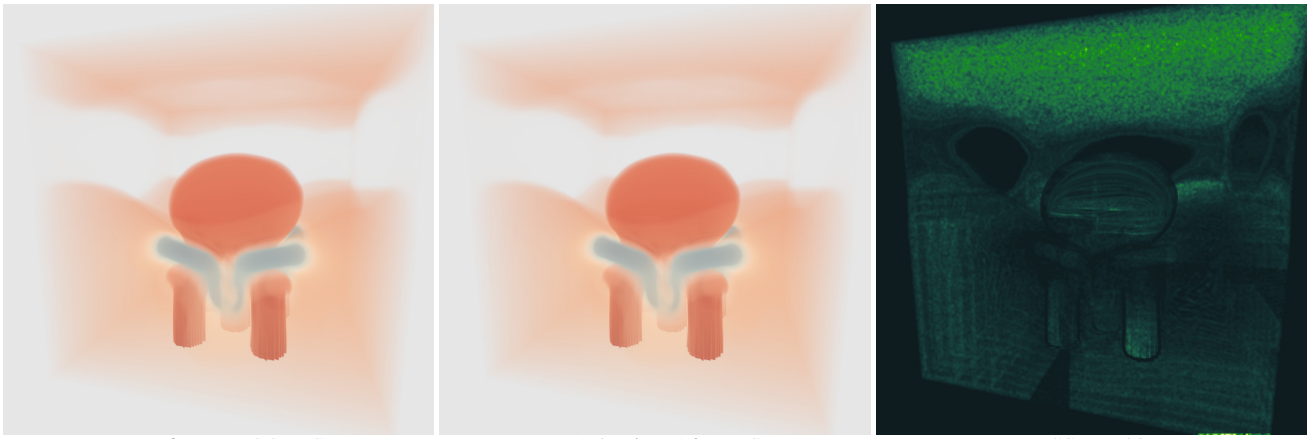
[32] I. Wald, W. Usher, N. Morrical, L. Lediaev, and V. Pascucci. RTX Beyond Ray Tracing: Exploring the Use of Hardware Ray Tracing Cores for Tet-Mesh Point Location. In *Proceedings of High Performance Graphics*, 2019. (To Appear), `http://www.sci.utah.edu/~wald/Publications/2019/rtxPointQueries/rtxPointQueries.pdf`.

[33] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Transactions on Graphics*, 2014.

[34] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based ray casting for tetrahedral meshes. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, 2003.

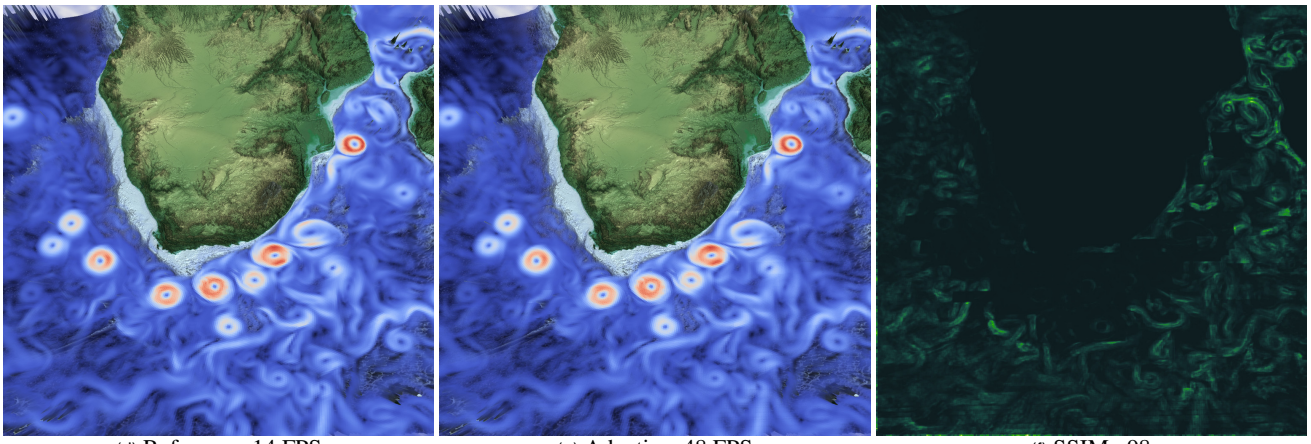
[35] K. Zimmermann, R. Westermann, T. Ertl, C. Hansen, and M. Weiler. Level-of-Detail Volume Rendering via 3D Textures. In *2000 IEEE Symposium on Volume Visualization (VV 2000)*, 2000.

Jets, 12M tets (vertex centered data)



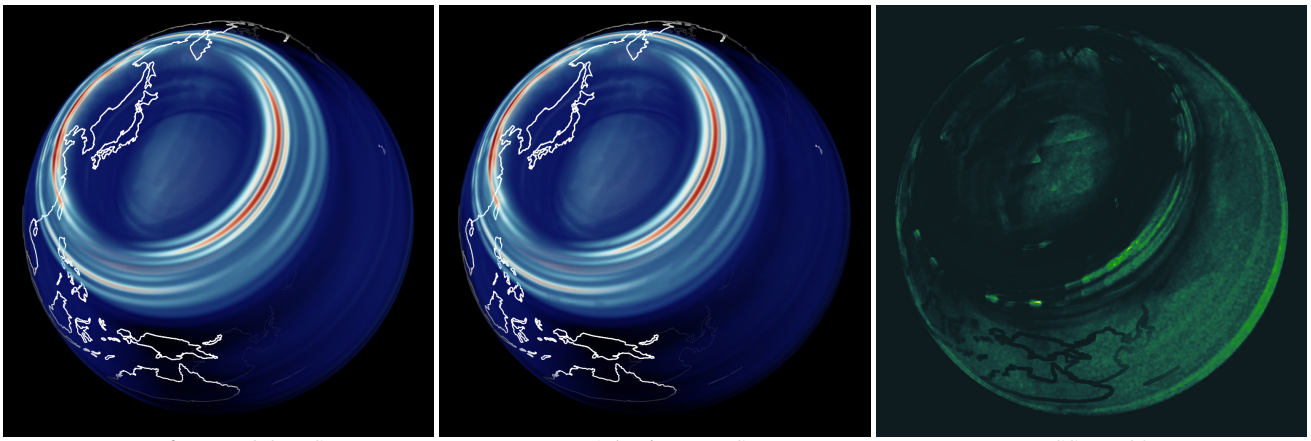
(a) Reference, 4.8 FPS (b) Adaptive, 16.7 FPS (c) SSIM, .997

Agulhas Current, 35M tets (cell centered data)



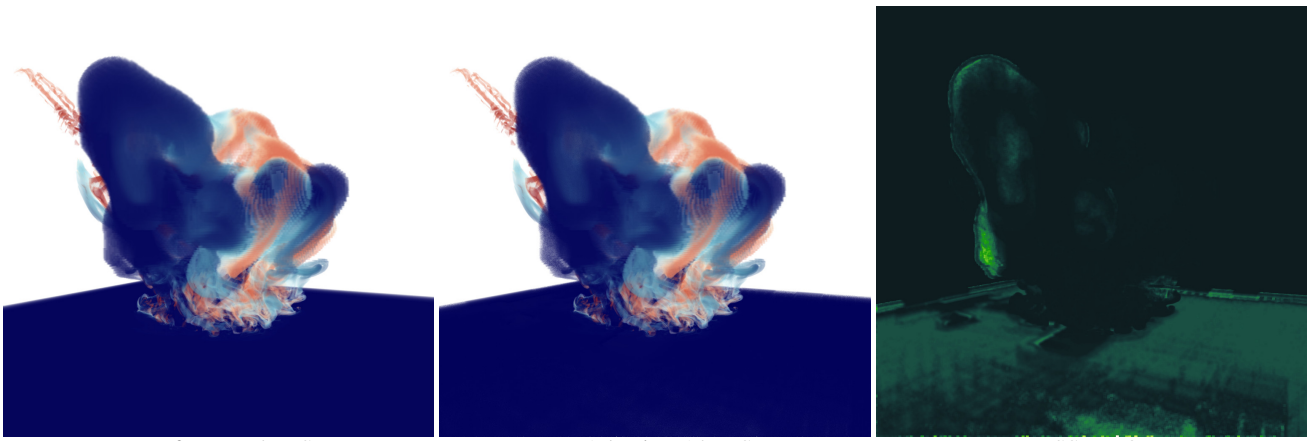
(d) Reference, 14 FPS (e) Adaptive, 48 FPS (f) SSIM, .98

Japan Earthquake, 278M tets (vertex centered data)



(g) Reference, 0.9 FPS (h) Adaptive, 7 FPS (i) SSIM, .98

Deep Water Asteroid Impact, 366M tets (cell centered data)



(j) Reference, 4 FPS (k) Adaptive, 14 FPS (l) SSIM, .97