A Visual Profiling System for Direct Volume Rendering

Max von Buelow[1], Daniel Ströter[1], Arne Rak[1] and Dieter W. Fellner[1,2]

[1]Technical University of Darmstadt, Germany [2]Fraunhofer IGD, Germany and Graz University of Technology, CGV Institute, Austria

Abstract

Direct Volume Rendering (DVR) is a crucial technique that enables interactive exploration of results from scientific computing or computer graphics. Its applications range from virtual prototyping for product design to computer-aided diagnosis in medicine. Although there are many existing DVR optimizations, they do not provide a thorough analysis of memory-specific hardware behavior. This paper introduces a profiling toolkit that enables the extraction of performance metrics, such as cache hit rates and branching, from a compiled GPU-based DVR application. The metrics are visualized in the image domain to facilitate spatial visual analysis. This paper presents a pipeline that automatically extracts memory traces using binary instrumentation, simulates the GPU memory subsystem, and models DVR-specific functionality within it. The profiler is demonstrated using the Octree-Linear Bounding Volume Hierarchy (OLBVH), and the visualized profiling metrics are explained based on the OLBVH implementation. Our discussion demonstrates that optimizing ray traversal for adaptive sampling, cache usage, branching, and global memory access has the potential to improve performance.

CCS Concepts

• ***Software and its engineering*** $\rightarrow$ ***Massively parallel systems;*** • ***General and reference*** $\rightarrow$ ***Performance;*** • ***Human-centered computing*** $\rightarrow$ *Visualization toolkits;*

1. Introduction

Many tasks in scientific computing or virtual prototyping include numerical simulations. Due to its robustness, unstructured tetrahedral meshing is frequently used for simulations. The simulation results are assigned to each tetrahedron or vertex, representing a scalar field. Post-processing uses DVR to enable exploration of simulation results. For interactive exploration, a ray marching (see fig. 1) implementation using the impressive aggregated parallel processing power of a graphics processing unit (GPU) is necessary. Many GPU-based DVR approaches emerged over the past decades, steadily improving performance. Only little work has been invested into DVR-specific profiling tools, which allow for the illumination of bottlenecks. We present an efficient and easy to use approach to facilitate profiling DVR using GPU binary instrumentation [VSNK19] and a visual encoding of extracted metrics. In summary, our **contributions** are:

- An extensible visual profiling system to visualize cache hit rates and code branching in the spatial domain of the frame buffer.
- An analysis of several configurations of a state-of-the-art DVR implementations using our proposed profiler.

1.1. Related Work

Typically, DVR methods use a spatial data structure to quickly confine the potentially intersecting primitives [MOB*21]. As performance benefits from GPU-parallel construction and traversal,

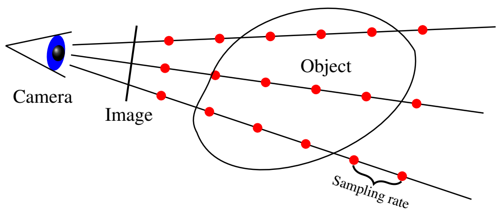

Figure 1: *In DVR, the camera emits view rays through the object. The scalar field is sampled at sampling points (red) along the rays.*

the linear BVH [LGS*09] (LBVH) uses a space-filling MORTON curve. Many authors evolved the LBVH addressing other types of spatial data structures [Kar12], better quality of bounding volumes [DP15], and faster traversal [GMOR14]. As GPU-memory is limited, sparse use of memory is another important property of a BVH. Memory-efficient rendering of large unstructured volume data is ongoing research [ZWMW23].

Performance improvements of unstructured DVR are achieved by reducing samples using adaptive sampling and empty space skipping [MUWP19]. To reduce memory consumption while still providing random-access point sampling, the mesh and spatial data structure can be compressed using HILBERT curves [WMZ22]. The OLBVH [SMSF20] achieves memory-efficient DVR building upon LBVH without using RTX-hardware and enables empty space skipping and reduction of the hierarchical tree depth to reduce memory

DOI: 10.2312/egs.20241030

consumption. As no specification of RTX-accelerated rendering is available, we use the OLBVH in this paper.

Performance visualization is an active research area with individual approaches [IGJ*14]. DVR applications render the visual output of simulation results as images, making them a natural choice for inter-domain mapping [SLB*11]. Our basis is that we map performance data of the hardware domain to the 2D grid [YBD01] structured frame buffer of the ray marcher, which resides a software domain. While standard profiling applications such as NVIDIA Nsight Compute can provide advanced insights into computational behavior, they are limited by the scope of a kernel call, which is too coarse to be used for application domain visualization. To overcome this issue, dynamic binary instrumentation can be used in order to capture hardware metrics for visualization purposes [vBRGF22].

While there are many papers about improving DVR performance, only little work deeply analyse performance on a fine-grained level. In this paper, we present a technique to visually guide efficient implementation of DVR approaches using profiling metrics mapped onto the domain of the framebuffer.

2. Visual Profiler for DVR

Our pipeline is based on a toolkit [BGF22] that provides a simulator of a typical memory pipeline of an NVIDIA GPU. The simulation is necessary to address the issue that cache hit rates are only provided on a per-kernel basis when using standard profilers such as NVIDIA Nsight Compute. The simulation automatically exracts a list of *memory requests* during the running time of a kernel and performs address coalescing resulting in a list of *memory transactions*. The metrics described below use the resulting cache hit rate annotations on a per memory transaction basis.

2.1. Cache Hit Rates

For cache hit rate visualization, we use this list of annotated memory transactions and group cache hit rates with respect to the grid coordinates (x,y) and allocation in order to transform the flat list of these annotations into the spatial domain of the output image. For a more detailed overview into the program behavior, we do this aggregation for each memory allocation seperately. Our toolkit automatically identifies and separates individual memory allocations of the profiled program and assigns each memory access a continuous allocation identifier i resulting in multiple sets of memory transactions $a_{(x,y,i)}$. This allows us to use a simple array of two-dimensional hit rate counters to handle the desired accumulation. The operator $\lceil\cdot\rceil$ denotes the number of hits in a set of memory transactions. The hit rate is calculated for each pixel as follows:

$$p_{(x,y,i)}(hit) = \frac{\lceil a_{(x,y,i)} \rceil}{|a_{(x,y,i)}|}. \tag{1}$$

These values $p_{(x,y,i)}(hit) \in [0,1]$ are then transformed to color values using standard color mapping.

2.2. Branching and Number of Memory Accesses

The general idea of our branching metric in the image domain is similar to the cache hit rate accounting. As GPUs mainly follow the SIMD model [Fly66], branching is implemented by executing each individual *diverged* control path in sequential passes. Each of these sequentialized control paths must deactivate corresponding diverged threads. Therefore, it is generally less efficient to have a high factor of branching in a GPU program. In order to measure divergence or branching, we allocate a counter for each pixel and increment whether a pixel mapped to a thread within a warp was active within that warp or not. The binary instrumentation tool *NVBIT* [VSNK19] allows extraction of this information using the instruction predicate that uses a 32 bit integer in order to bit-wise encode if each of the $w = 32$ threads within a warp was active or not. As this paper focuses solely on memory instructions, we limit visual profiling to memory operations, similar to the visualization of cache hit rates. We denote the number of active threads within a warp i as n_i and the set of warp-wise memory requests per pixel as $r_{(x,y)}$. We define the number of memory requests per pixel $N_{(x,y)}$ and the branching rate $b_{(x,y)} \in [0,1]$ as follows:

$$b_{(x,y)} = \frac{N_{(x,y)}}{w \cdot |r_{(x,y)}|}, \quad N_{(x,y)} = \sum_{i \in r_{(x,y)}} n_i. \tag{2}$$

(a) *SimJEB122* **(b)** *SimJEB514*

Figure 2: *Render images of the used SimJEB models.*

2.3. Visualization System Requirements

We target experienced software engineers in the computer graphics and ray tracing domains, who try to optimize their ray tracing application in terms of run-time performance. Standard profilers only profile coarse metrics at the granularity of a kernel call. Our visual profiler should enable further insight into performance data mapped to the spatial domain of the ray marching result, which potentially helps to find inefficiencies in spatial data structures. This leads us to the following requirements for our visualization:

(R1) **Visualization**: The user should be able to see performance data mapped onto the frame buffer of the DVR application for inspecting the performance characteristics of the program.

(R2) **Analysis**: The user should be able to compare with visualizations of other programs or configurations to find potential bottlenecks in one single visualization.

3. Discussion

We discuss general observations in section 3.1 and visualized performance metrics in section 3.2, respectively. We demonstrate the functionality of our profiling tool using meshes of the SimJeb dataset [WBM21] that includes optimized mechanical brackets from an engineering design competition. The OLBVH construction subtracts a specified number α from the maximal hierarchical tree depth to reduce memory consumption compressing the tree [SMSF20]. We choose the average edge length of a mesh as the reference sampling rate. While we tested our toolkit on many

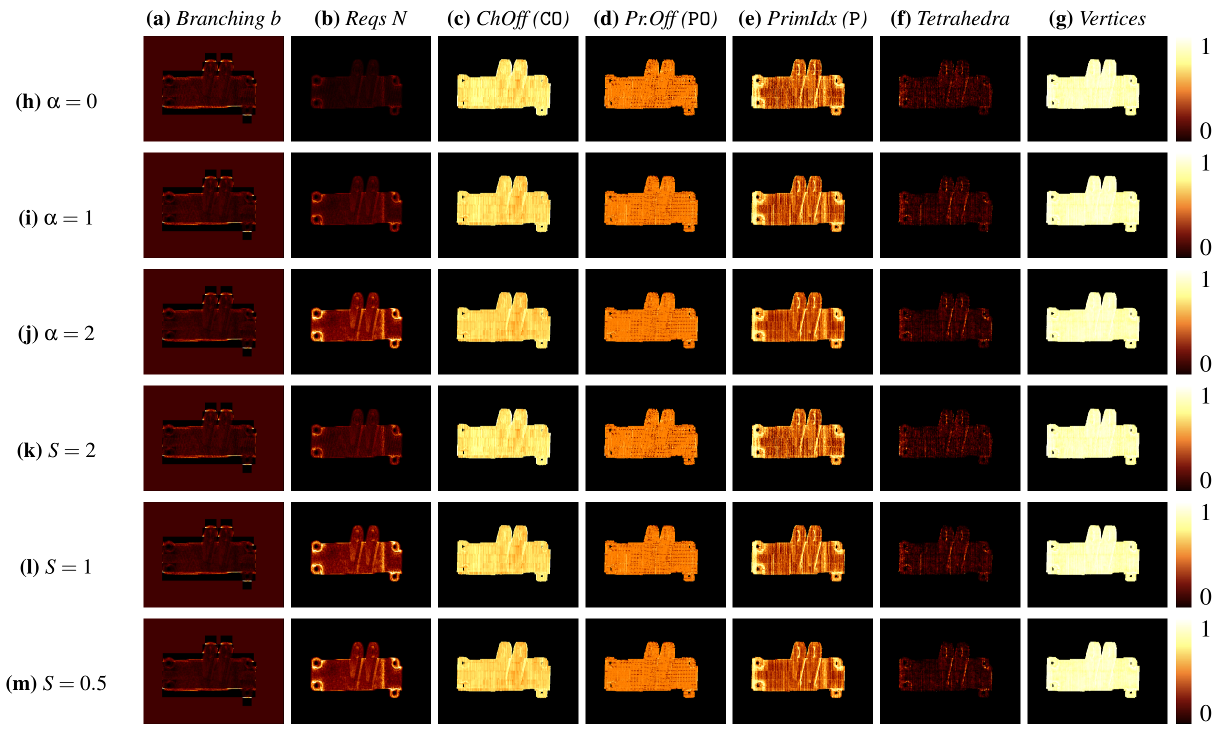

Figure 3: *Analysis for rendering the SimJeb514 model visualizing the branching rate $b_{(x,y)}$, numbers of requests $N_{(x,y)}$ normalized by 2^{15}, and cache hit rates $p_{(x,y,i)}$ for individual allocations i.*

models, we showcase the evaluation on the SimJEB514 with many sharp features and the SimJEB122 with many rounded features (see fig. 2). As the evaluation results of the two models mostly provide analog implications, the discussion focuses on the SimJEB514 model (see fig. 3), while the evaluation of the SimJEB122 model is part of the supplemental material.

Our evaluation covers several compression levels $\alpha \in \{0,1,2\}$ and factors the reference sampling rate by $S \in \{2,1,0.5\}$. We consistently used regular sampling in our experiments. The evaluation determines branching and counts global memory accesses denoted as *Reqs*. We determine the cache hit rates for particular array buffers related to traversal, in order to provide an in-depth analysis. For traversal, the most relevant arrays are the children offsets `CO` referring to the children of a tree node, primitive offsets `PO` referring to the tetrahedra of a tree node, and the primitive indices `P` referring to the individual tetrahedra. In addition, we also evaluate the caching of the tetrahedra and vertices of the mesh.

3.1. General Observations

Our visualizations in fig. 3 show trends (R1) that we analyze (R2). Each image shows the original structure of the model [SLB*11]. Another observation is that increasing the number of memory requests to a location does not necessarily improve cache hit rates, other than intuition might suggest. This is particularly evident observing the `CO` array in fig. 3c. As more frequent accesses to other allocations affect the caching of these allocations, increasing global memory accesses does not improve cache hit rates. The general trend is that vertices receive the best cache hit rates, which we explain with the size of contiguous read transactions.

3.2. Performance Visualization

The visualization in fig. 3 enables performance analysis (R2) for varying compression and sampling rates for rendering the SimJEB514 model. Most branching occurs for the rays intersecting only with the silhouette boundary of the model. Thus, these rays reduce the occupancy rate. Using compression or finer sampling rates does not lead to significant differences in regards to branching behavior. On the contrary, the number of global accesses (fig. 3b) significantly increases when increasing α or reducing S. Global access numbers are especially large at the curved features of the model, where a meshing tool typically uses a finer resolution to represent the features with sufficient element quality for simulation.

Cache hit rates for the children offsets (`CO`) slightly increase for a shallower tree and a finer sampling rate, because traversal more often terminates in the same or memory-adjacent leaf nodes. However, hit rates for primitive offsets (`PO`) are equally distributed and do not significantly change for altering α or S. A more varying cache hit rate occurs for the primitive indices (`P`), while changing α or S does not exhibit significant influence. The cache hit rate for primitive indices is more governed by the scalar data and the structure of the model. As scalar values are low at the round holes, these regions are rendered with high transparency leading the DVR to spend more sampling points on the corresponding rays. Due to the round structure of the holes, the corresponding tree nodes contain more empty space and are associated with fewer tetrahedra, which leads to better cache hit rates.

One vertical line in the right parts of the images indicates a large number of accesses. The mesh resolution is finer along this vertical line leading to more memory accesses. Our visual representation additionally shows that cache hit rates increase at these positions. The distribution of cache hit rates for vertices is uniform for all choices of α or S for rendering both models.

4. Conclusion

Performance evaluation is an important aspect for understanding state-of-the-art DVR implementations and potential further improvements. We presented a visualization approach for GPU DVR ray marchers to visualize performance metrics in the spatial domain of the rendered image. We demonstrated our visualization using the OLBVH and explained the metrics based on its memory layout.

Our discussion shows that potential for improving performance lies in using adaptive sampling of the scalar field. In addition, varying the resolution in a mesh leads to an anisotropic distribution of global accesses, which is not inherently compensated by cache usage. The surface features of a mesh typically account for peaks in branching and memory accesses.

Future Work We plan to provide interactive profiling results of arbitrary DVR applications. We intend to support non-NVIDIA GPUs, which will require the use of other binary instrumentation tools. Finally, the branching metric that is currently limited to memory instructions could be extended to include arbitrary instructions.

Source Code and Acknowledgements The source code is available at `https://github.com/maxvonbuelow/insituprof`. We thank Armin Pauli for his work on the implementation. Additionally, we thank the anonymous reviewers whose comments helped us to improve our paper.

References

[BGF22] BUELOW, MAX VON, GUTHE, STEFAN, and FELLNER, DIETER W. "Fine-Grained Memory Profiling of GPGPU Kernels". *Computer Graphics Forum* 41.7 (Oct. 2022): *Pacific Graphics*, 227–235. DOI: 10.1111/cgf.14671 2.

[DP15] DOMINGUES, LEONARDO R. and PEDRINI, HELIO. "Bounding volume hierarchy optimization through agglomerative treelet restructuring". *Proceedings of the 7th Conference on High-Performance Graphics*. ACM, Aug. 7, 2015. DOI: 10.1145/2790060.2790065 1.

[Fly66] FLYNN, M.J. "Very high-speed computing systems". *Proceedings of the IEEE* 54.12 (1966), 1901–1909. DOI: 10.1109/proc.1966.5273 2.

[GMOR14] GARCÍA, ARTURO, MURGUIA, SERGIO, OLIVARES, ULISES, and RAMOS, FÉLIX F. "Fast parallel construction of stack-less complete LBVH trees with efficient bit-trail traversal for ray tracing". *Proceedings of the 13th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*. ACM, Nov. 30, 2014. DOI: 10.1145/2670473.2670488 1.

[IGJ*14] ISAACS, KATHERINE E., GIMÉNEZ, ALFREDO, JUSUFI, ILIR, et al. "State of the Art of Performance Visualization". *EuroVis - STARs*. The Eurographics Association, 2014. DOI: 10.2312/EUROVISSTAR.20141177 2.

[Kar12] KARRAS, TERO. "Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees". en. *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. The Eurographics Association, 2012. DOI: 10.2312/EGGH/HPG12/033-037 1.

[LGS*09] LAUTERBACH, C., GARLAND, M., SENGUPTA, S., et al. "Fast BVH Construction on GPUs". *Computer Graphics Forum* 28.2 (Mar. 27, 2009), 375–384. DOI: 10.1111/j.1467-8659.2009.01377.x 1.

[MOB*21] MEISTER, DANIEL, OGAKI, SHINJI, BENTHIN, CARSTEN, et al. "A Survey on Bounding Volume Hierarchies for Ray Tracing". *Computer Graphics Forum* 40.2 (May 2021), 683–712. DOI: 10.1111/cgf.142662 1.

[MUWP19] MORRICAL, NATE, USHER, WILL, WALD, INGO, and PASCUCCI, VALERIO. "Efficient Space Skipping and Adaptive Sampling of Unstructured Volumes Using Hardware Accelerated Ray Tracing". *2019 IEEE Visualization Conference (VIS)*. IEEE, Oct. 2019. DOI: 10.1109/visual.2019.8933539 1.

[SLB*11] SCHULZ, MARTIN, LEVINE, JOSHUA A., BREMER, PEER-TIMO, et al. "Interpreting Performance Data across Intuitive Domains". *2011 International Conference on Parallel Processing*. IEEE, Sept. 2011, 206–215. DOI: 10.1109/icpp.2011.60 2, 3.

[SMSF20] STRÖTER, DANIEL, MUELLER-ROEMER, JOHANNES S., STORK, ANDRÉ, and FELLNER, DIETER W. "OLBVH: octree linear bounding volume hierarchy for volumetric meshes". en. *The Visual Computer* 36.10-12 (July 6, 2020), 2327–2340. DOI: 10.1007/s00371-020-01886-6 1, 2.

[vBRGF22] VON BUELOW, MAX, RIEMANN, KAI, GUTHE, STEFAN, and FELLNER, DIETER W. "Profiling and Visualizing GPU Memory Access and Cache Behavior of Ray Tracers". *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2022, 7–17. DOI: 10.2312/PGV.20221061 2.

[VSNK19] VILLA, ORESTE, STEPHENSON, MARK, NELLANS, DAVID, and KECKLER, STEPHEN W. "NVBit. A Dynamic Binary Instrumentation Framework for NVIDIA GPUs". *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '52. ACM, Oct. 12, 2019. DOI: 10.1145/3352460.3358307 1, 2.

[WBM21] WHALEN, E., BEYENE, A., and MUELLER, C. "SimJEB: Simulated Jet Engine Bracket Dataset". *Computer Graphics Forum* 40.5 (Aug. 2021), 9–17. DOI: 10.1111/cgf.14353 2.

[WMZ22] WALD, INGO, MORRICAL, NATE, and ZELLMANN, STEFAN. "A Memory Efficient Encoding for Ray Tracing Large Unstructured Data". *IEEE Transactions on Visualization and Computer Graphics* 28.1 (Jan. 2022), 583–592. DOI: 10.1109/tvcg.2021.3114869 1.

[YBD01] YU, Y., BEYLS, K., and D'HOLLANDER, E.H. "Visualizing the impact of the cache on program execution". *Proceedings Fifth International Conference on Information Visualisation*. IEEE Comput. Soc, 2001, 336–341. DOI: 10.1109/iv.2001.942079 2.

[ZWMW23] ZELLMANN, STEFAN, WU, QI, MA, KWAN-LIU, and WALD, INGO. "Memory-Efficient GPU Volume Path Tracing of AMR Data Using the Dual Mesh". *Computer Graphics Forum* 42.3 (June 2023), 51–62. DOI: 10.1111/cgf.14811 1.