Eurographics Symposium on Parallel Graphics and Visualization (2021)
M. Hadwiger, M. Larsen, F. Sadlo (Editors)

Supplemental Material for "Evaluation of PyTorch as a Data-Parallel Programming API for GPU Volume Rendering"

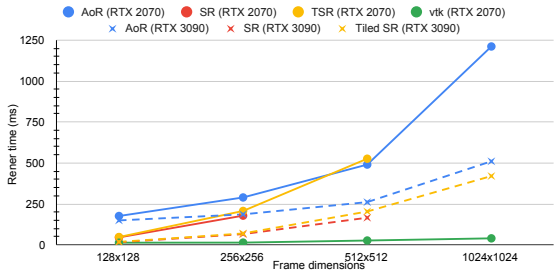

Figure 1: *Output resolution vs. render time for the* 512^3 *Rayleigh-Taylor instability volume. SR results could not be recorded at* 512^2 *and above on the RTX 2070 due to memory usage exceeding 8GB. For the same reason, TSR results could not be recorded at* 1024^2 *on the 2070. VTK times (all on the 2070) are between 13-40 ms.*

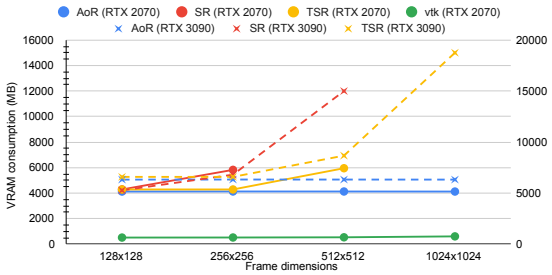

Figure 2: *Output resolution versus memory usage. See Figure 1 caption for benchmark parameters, which are the same here. VTK memory consumption ranged between 523-616 MB. Even the most memory-efficient PyTorch-based approaches consume about an order of magnitude more VRAM, highlighting limited scalability.*

1. Implementation details for Algorithm 1 ("Array of Rays")

Assume a framebuffer with dimensions $m \times n$, and assume exactly one ray is cast through each pixel. Let each ray's origin be $\mathbf{e} \in \mathbb{R}^3$, which is the eye position. Let the 3D array $D \in \mathbb{R}^{m \times n \times 3}$ store the direction of each ray, as determined by the pinhole camera model. (D can be thought of as a 2D array, each element of which is a 3D unit vector that stores a direction.)

Using $\mathbf{e}$ and D, we intersect each ray with the volume AABB, and we obtain the 2D arrays $T_{min}, T_{max} \in \mathbb{R}^{m \times n}$, which have the same dimensions as D, and hold the minimum and maximum hit distances for each ray. The ray-box intersection was performed by following Lombardi et al.'s approach [LSS*19], which uses vectorized arithmetic and comparison operations, in conjunction with Blelloch's "`p-select`" operation [Ble90], which is available in PyTorch as `torch.where`.

We use the array $T_{eval} \in \mathbb{R}^{m \times n}$ to store the distance at which we evaluate each ray, and initialize this to T_{min}. We raymarch by advancing T_{eval}, evaluating each ray, and storing the 3D sample positions in $P \in \mathbb{R}^{m \times n \times 3}$. Using the (structured) volume sampling function built into PyTorch, we find the g_i values (emissive contribution, see Section 3.2) at positions P and store them in $G \in \mathbb{R}^{m \times n}$. Similarly, the alpha values at positions P are stored in $A \in \mathbb{R}^{m \times n}$. We apply a vectorized "over" operator using elementwise arithmetic, and store the result in $L \in \mathbb{R}^{m \times n}$.

Finally, R_{done} is a $m \times n$ logical array that keeps track of which rays are "finished". R_{done} is initialized to all zeros (the zero matrix is denoted by $\mathbb{0}^{m \times n}$), and the rendering loop terminates when every element of R_{done} is equal to 1.

2. Additional Benchmarks

Benchmark results for fixed volume size (512^3) and varying resolution, all on the Rayleigh-Taylor dataset, are shown in Figures 2 and 1. Profiling results for a different dataset ("Magnetic Reconnection") are shown in Figure 3.

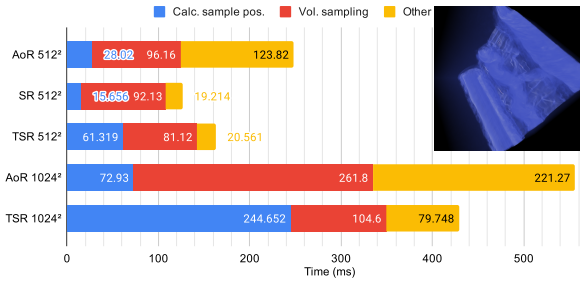

Figure 3: *Profiling results on the RTX3090 for the* 512^3 `float32` *magnetic reconnection dataset [GLDL14], upper right.*

References

[Ble90] BLELLOCH G.: *Vector Models for Data-parallel Computing*. AI Series. MIT Press, 1990. 1

[GLDL14] GUO F., LI H., DAUGHTON W., LIU Y.-H.: Formation of hard power laws in the energetic particle spectra resulting from relativistic magnetic reconnection. *Phys. Rev. Lett. 113* (Oct 2014), 155005. 1

[LSS*19] LOMBARDI S., SIMON T., SARAGIH J., SCHWARTZ G., LEHRMANN A., SHEIKH Y.: Neural volumes: Learning dynamic renderable volumes from images. *ACM Trans. Graph. 38*, 4 (July 2019), 65:1–65:14. 1