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

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 \( \mathbf{D} \in \mathbb{R}^{m \times n \times 3} \) store the direction of each ray, as determined by the pinhole camera model. \( \mathbf{D} \) 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 \( \mathbf{D} \), we intersect each ray with the volume AABB, and we obtain the 2D arrays \( \mathbf{T}_{\text{min}}, \mathbf{T}_{\text{max}} \in \mathbb{R}^{m \times n} \), which have the same dimensions as \( \mathbf{D} \), 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 \( \text{torch.where} \).

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

Finally, \( \mathbf{R}_{\text{done}} \) is a \( m \times n \) logical array that keeps track of which rays are “finished”. \( \mathbf{R}_{\text{done}} \) is initialized to all zeros (the zero matrix is denoted by \( \mathbf{0}^{m \times n} \)), and the rendering loop terminates when every element of \( \mathbf{R}_{\text{done}} \) 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.

© 2021 The Author(s)
References

