CUDA-Accelerated RF-NeRF: Fast Volumetric Rendering with RF-Conditioned Fields

Benjamin J. Gilbert

Spectrcyde RF Quantum SCYTHE, College of the Mainland
bgilbert2@com.edu
ORCID: https://orcid.org/0009-0006-2298-6538

TABLE I
SUMMARY (TOP CONFIGS).

Device	Samples	Chunk	Rand.	ms/frame	rays/s	PSNR
cpu	128.000	32 768.000	no	3649.900	10 521.000	99.000
cpu	128.000	8192.000	no	771.100	49 801.000	97.600
cpu	128.000	32 768.000	no	864.100	44 438.000	97.600
cpu	64.000	32 768.000	no	380.600	100 880.000	85.600
cpu	64.000	8192.000	no	384.900	99 778.000	85.600

***Abstract*—We present a CUDA-accelerated renderer for RF-conditioned NeRF with GPU kernels for ray generation, stratified sampling, and volumetric integration. A small benchmark sweeps samples and chunk sizes, logging PSNR/SSIM vs. latency; JSON→LaTeX keeps tables and plots reproducible.**

Reproducibility: commit f2017942, seed 42, device cpu, built 2025-09-13 05:58:59 CEST.

I. Introduction

We target real-time RF scene rendering by moving NeRF ray operations to GPU kernels. Our renderer (`code/cuda_nerf_renderer.py`) provides kernels for ray casting, point sampling, and transmittance integration.

II. Method

RF-Conditioned Field. The NeRF MLP takes $(\mathbf{x} \in \mathbb{R}^3)$ and an RF feature vector $\mathbf{r} \in \mathbb{R}^{d_r}$ and outputs color $\mathbf{c} \in \mathbb{R}^3$ and density $\sigma \in \mathbb{R}_{\geq 0}$.

CUDA Kernels. We implement: (i) ray generation; (ii) stratified sampling along rays; (iii) volumetric rendering with early termination.

III. Evaluation

We synthesize a reference image (high samples) and compare PSNR/SSIM vs. runtime for various sample counts/chunk sizes. Scripts produce figures and siunitx tables.

IV. Related Work

NeRF [1] introduced radiance fields; real-time variants exploit GPU acceleration and hash encodings [2]. We use classic volume rendering with transmittance. For Python GPU, we rely on Numba CUDA. SSIM follows [3].

TABLE II
ABLATIONS (SAMPLES / CHUNK / RANDOMIZED).

Device	Samples	Chunk	Rand.	ms/frame	rays/s	PSNR	SSIM
cpu (ref)	128.000	32 768.000	no	3649.900	10 521.000	99.000	1.000
cpu	32.000	8192.000	yes	202.700	189 473.000	21.000	0.617
cpu	32.000	8192.000	no	201.300	190 763.000	72.800	1.000
cpu	32.000	32 768.000	yes	196.000	195 941.000	21.100	0.637
cpu	32.000	32 768.000	no	261.900	146 594.000	72.800	1.000
cpu	64.000	8192.000	yes	416.000	92 308.000	20.900	0.616
cpu	64.000	8192.000	no	384.900	99 778.000	85.600	1.000
cpu	64.000	32 768.000	yes	393.000	97 701.000	21.200	0.637
cpu	64.000	32 768.000	no	380.600	100 880.000	85.600	1.000
cpu	128.000	8192.000	yes	813.000	47 234.000	21.000	0.623
cpu	128.000	8192.000	no	771.100	49 801.000	97.600	1.000
cpu	128.000	32 768.000	yes	864.000	44 442.000	21.000	0.618
cpu	128.000	32 768.000	no	864.100	44 438.000	97.600	1.000

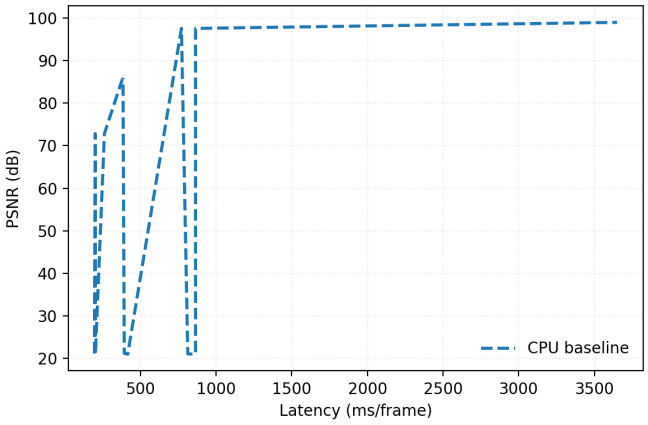

Fig. 1. Pareto: PSNR vs. latency; dashed line denotes CPU baseline.

V. Reproducibility

Run `make -f Makefile_nerf pdf`. If CUDA is unavailable, the bench gracefully falls back to a NumPy/torch CPU renderer, still producing metrics, tables, and figures.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020.

[2] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," in *SIGGRAPH*, 2022.

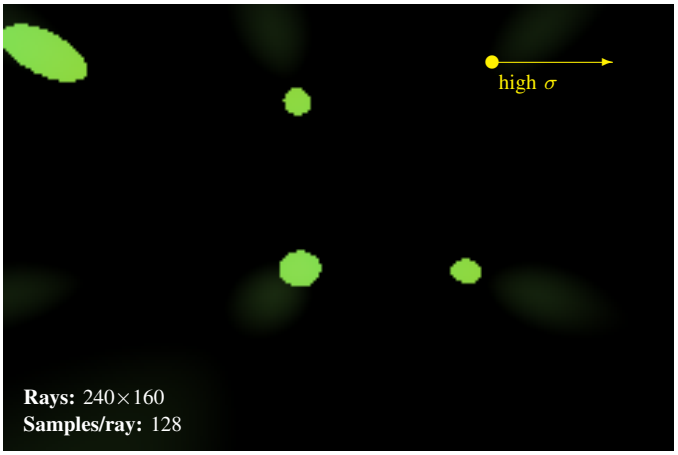

Fig. 2. Reference frame with vector overlay (Times font).

[3] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE TIP*, 2004.