Report

Chin-Hui Chu r12944041

1. Implementation Overview

- **CUDA structure.** The renderer launches three kernels per frame: `traceKernel` marches rays and records hit data, `shadeKernel` computes lighting for each sample, and `finalizeKernel` averages the per-sample colors into the final RGBA image. Host code allocates device buffers for distances, traps, directions, hits, and sample colors, then dispatches the kernels sequentially before copying the image back.
- **Thread/block partitioning.** Pixels are supersampled with AA=3 (`9` samples). Each CUDA thread processes exactly one sample, so a warp walks contiguous samples. We use a fixed `kSampleBlockSize = 128` threads per block; the grid is sized to cover `width * height * AA^2` samples for tracing/shading and `width * height` pixels for the finalize pass.
- **Optimizations applied.**
  - *Distance estimator fast math:* The Mandelbulb distance estimator now uses `float` transcendentals (`atan2f` , `asinf` , `powf` , `sincosf`) with casts back to double, slashing the latency chain in both trace and shading.
  - *Shared DE helper:* `mandelbulbDEInternal` and `mapInternal` expose a tunable iteration count. `calcNor` leverages this to reuse work and compute central differences with half the iterations, reducing the six extra DE replays per surface sample.
  - *Normal reuse in shading:* `shadeKernel` passes the marched distance into `calcNor` , avoiding an additional full `map` call for the base point.
  - *Restrictive launch bounds:* `__launch_bounds__(128,7)` keeps register usage low (47 registers in `traceKernel` , 60 in `shadeKernel`) without spills, ensuring consistent occupancy.
  - *Fast shading math:* Lighting is performed in `float3` , trimming register pressure and SFU usage compared to the double-precision CPU reference.

2. Performance Analysis

GPU kernel timing (nvprof)

Testcase	Resolution	`traceKernel` (ms)	`shadeKernel` (ms)
Case 00	64×64	8.81	3.17
Case 01	512×512	30.35	15.77

Observation: Fast-math and normal re-use reduce Case 00's trace pass from 25 ms (baseline) to ~9 ms. Case 01 remains compute-bound but benefits from higher throughput per warp.

Nsight Compute highlights

Testcase	`No Eligible` (%)	CPI Stall (exec dep)	Avg active threads/warp	Dominant pipeline
Case 00 (64×64)	67.6 (trace) / 70.4 (shade)	47–53 %	13 / 26	FP32/64 mix
Case 01 (512×512)	23.3 (trace) / 29.6 (shade)	42–46 %	13 / 26	FP32 FMA (~53 %)

Observations:

- Small grids (Case 00) are still limited by warp availability even after fast math; divergence in the distance estimator leaves only ~13 active lanes per warp.
- Larger grids (Case 01) achieve ~70 % issue-slot utilization; the workload becomes compute-limited with the FMA pipe saturated. Divergence remains the main inefficiency (normals/shadows), but scheduler starvation is no longer the bottleneck.
- L1/TEX hit rates fall to ~55 % after switching to float math, yet memory pipes remain <1 % busy, confirming these kernels are compute-heavy.

Block-size sweep (Case 00, 64×64)

Block Size	Grid Size	Issue Slots Busy (%)	`No Eligible` (%)	Warp Cycles / Issued Instr
64	576	20.62	66.64	4.54
128 (base)	288	19.96	67.82	4.52
256	144	20.85	67.00	4.55

Observation: Changing the launch block size does not materially affect utilization—issue slots stay near 20 %, `No Eligible` hovers around 67 %, and CPI remains ~4.5 cycles/instruction. Divergence dominates regardless of block geometry; larger blocks do not hide the latency further for this small workload.

Additional notes

- API overhead (six `cudaMalloc` + sync) still costs ~90 ms at 64×64 and ~140 ms at 512×512 (`profiles/*_timing.txt`). Moving to persistent allocations would help interactive use.
- Final color accumulation currently writes every supersample; for large frames this pushes >900 GB/s of device traffic. In-place accumulation or smaller data formats (`float4` or `half`) are attractive next steps.

3. Conclusion

- **Challenges.** Shrinking the normal estimation cost without visible artifacts was tricky—aggressive forward-difference shortcuts broke validation. Balancing accuracy with reduced iteration counts required careful profiling and incremental changes. The hardest outstanding problem is warp divergence in the marching and shading loops; fixing it demands structural work (work queues, compaction, adaptive stepping), not just a local tweak, so it remains future work.