Examination of the Nvidia RTX

Introduction

- RTX promises "Real time ray tracing"
- RT core accelerating ray tracing workloads
- Turing architecture whitepaper:
- Bounding box and ray triangle intersection units
- Techniques used in RTX hardware

Related ray tracing hardware

- First dedicated hardware example: Volumepro 1999
- Implemented ray grouping
- Memory coherence
- Fully pipelined SaarCOR 2003 (traversing, memory load, intersection)
- First programmable hardware RPU 2005
- TRaX (2008), MIMD
- Filtering important for SIMD to combat incoherent groups

2. Related ray tracing hardware

Architecture considerations for tracing incoherent rays 2010

- Believed that Fermi GPU (2010) included work compaction
- Partitioning BVH into treelets:
- Grouping rays to treelets they intersect

Other papers

- Avoiding random memory access during traversal (2017)
- MIMD using reduced precision BVH (2014)

Nvidia RTX

- Available through Vulkan, Microsoft DirectX 12 and Nvidia OptiX API's.
- API's expose a ray tracing pipeline for the developer
- Acceleration structures for developer
- Shader types: ray-generation, miss, closest hit and intersection
- Contains triangle intersection unit (Nvidia whitepaper)
- Triangle intersection unit 2-3.5 times faster than volta (software impl.)

Experiments

- Using GTX 1070 and RTX 2070
- Using two path tracers with Vulkan and DirectX and Hydra renderer
- Impl_1 Vulkan and Impl_2 DirectX
- Benchmark metric total rays:

$$rays = width * height * spp * fps$$

scene	primary	secondary	tertiary
Sponza, impl_1	807	437	806
Sponza, impl_2	928	777	694
Sponza, Hydra_SW	480	122	130
Crysponza, impl_1	806	419	388
Crysponza, impl_2	754	635	216
Crysponza, Hydra_SW	276	92	80
Hairballs, impl_1	275	223	256
Hairballs, impl_2	567	155	141
Hairballs, Hydra_SW	61	50	56

Table 1. Million rays traced per second (Mrays/s), 1 sample per pixel, 1024 x 1024 resolution, RTX2070

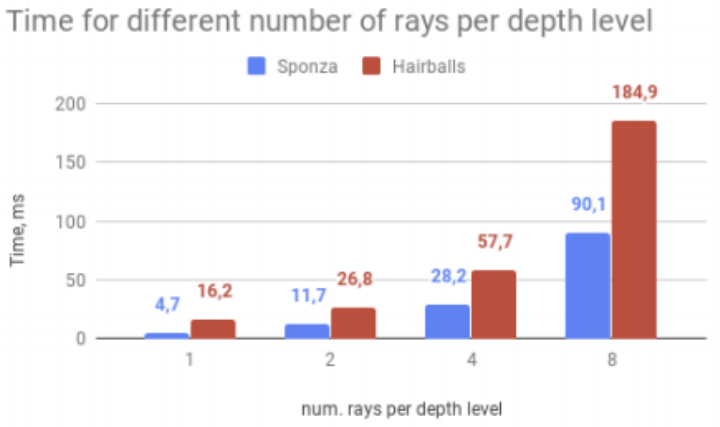


Fig. 1. Time spent by ray tracing "draw call" per frame (1 sample per pixel, 1024 x 1024 resolution) depending on rays traced per depth level. Depth = 3

Hydra vs. RTX (hardware and software impl.)

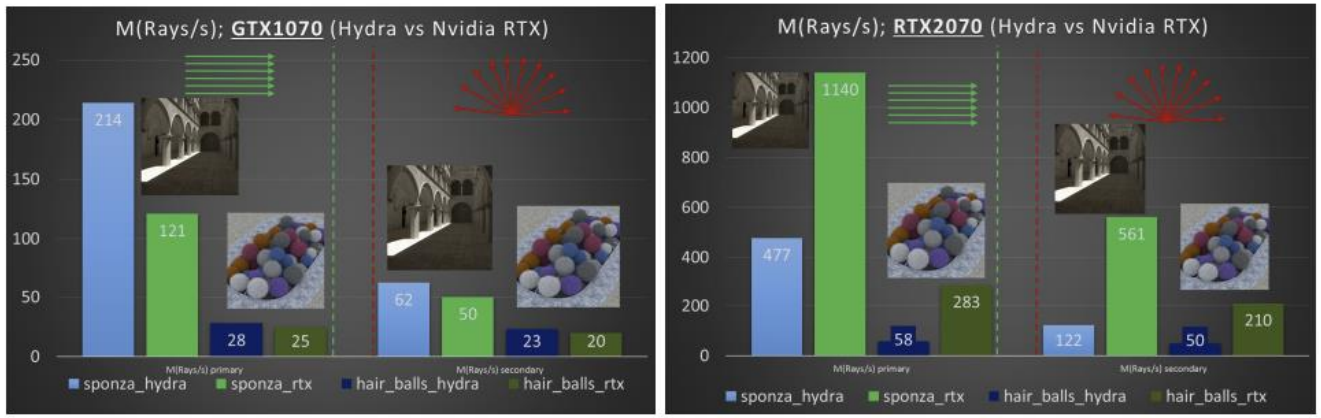


Fig. 2. Comparison on GTX1070 (left) and RTX2070 (right) (Open Source implementation vs Nvidia RTX). The left part of each image (green) shows performance for primary (coherent) rays, and the right part (red) for secondary (random) rays.

Sponza (66K tris) Cry Sponza (262K tris) Hair Balls (224M tris)

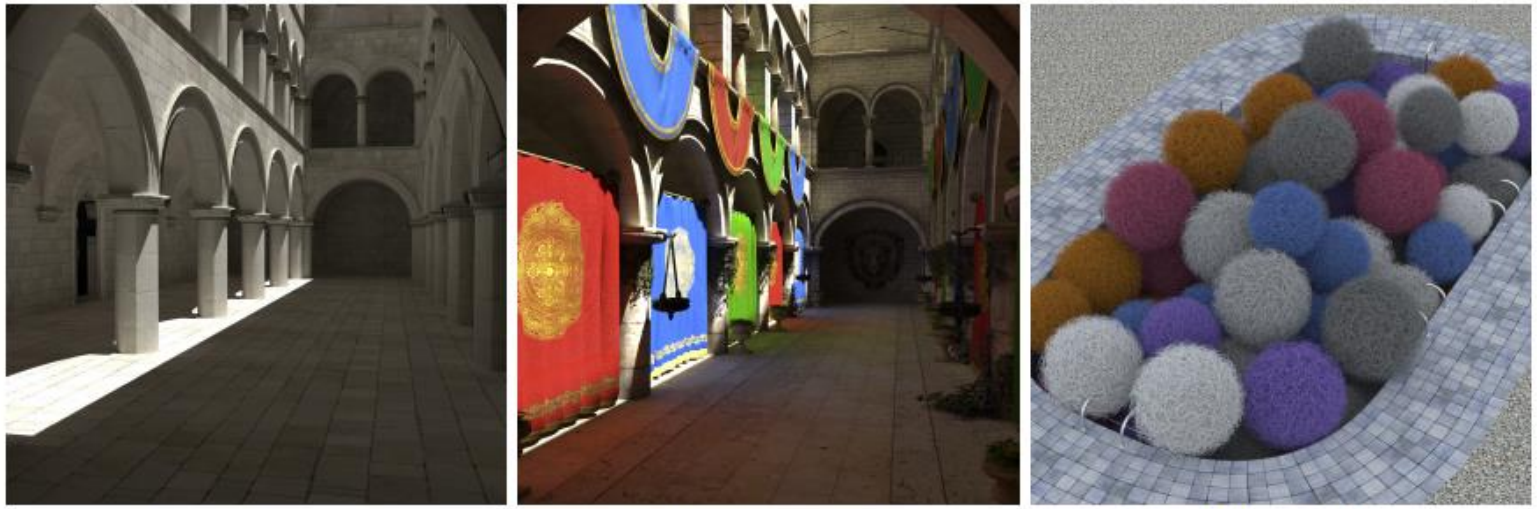
Fig. 3. Test Scenes

Conclusion 1

- Nvidia RTX accelerating random memory access during ray tracing
- Performance is 2x 1070 at coherent rays
- Performance is 4-5x 1070 at incoherent rays
- Bottleneck seems to be memory related at incoherent rays

Conclusion 2

- Nvidia RTX implements some ray-grouping/ray-sorting
- Hardware impl. Smaller performance drop on second ray
- Harder scene performance drop is similar



Sponza (66K tris) Cry Sponza (262K tris) Hair Balls (224M tris)

Conclusion 3

- CPU Kernel style progrmming is inefficient for ray tracing
- 1070 performs worse with RTX than with hydra renderer
- A more complicated kernel also reduces the performance a lot on second->3,4

scene	primary	secondary	tertiary
Sponza, impl_1	807	437	806
Sponza, impl_2	928	777	694
Sponza, Hydra_SW	480	122	130
Crysponza, impl_1	806	419	388
Crysponza, impl_2	754	635	216
Crysponza, Hydra_SW	276	92	80
Hairballs, impl_1	275	223	256
Hairballs, impl_2	567	155	141
Hairballs, Hydra_SW	61	50	56

Table 1. Million rays traced per second (Mrays/s), 1 sample per pixel, 1024 x 1024 resolution, RTX2070

Conclusion 4

- Nvidia are using GPU work creation for rays

Recursivly