Adding RTX acceleration to Iray with OptiX 7

Lutz Kettner Director Advanced Rendering and Materials

July 30th, SIGGRAPH 2019

What is Iray?

Production Rendering on CUDA

Bring ray tracing based production / simulation quality rendering to GPUs

New paradigm: ***Push Button*** rendering (open up new markets)

Plugins for


3ds Max


Maya


Rhino


SketchUp

...

In Production since > 10 Years

SIMULATION QUALITY

ARTISTIC FREEDOM

How Does it Work?

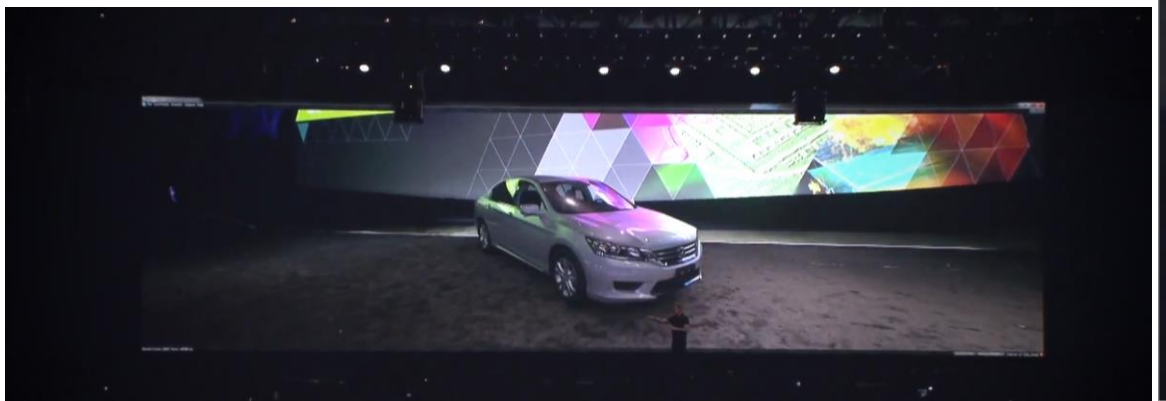
99% physically based Path Tracing

To guarantee simulation quality and ***Push Button***

- Limit shortcuts and good enough hacks to minimum
- Brute force (spectral) simulation

  no intermediate filtering

  scale over multiple GPUs and hosts even in interactive use

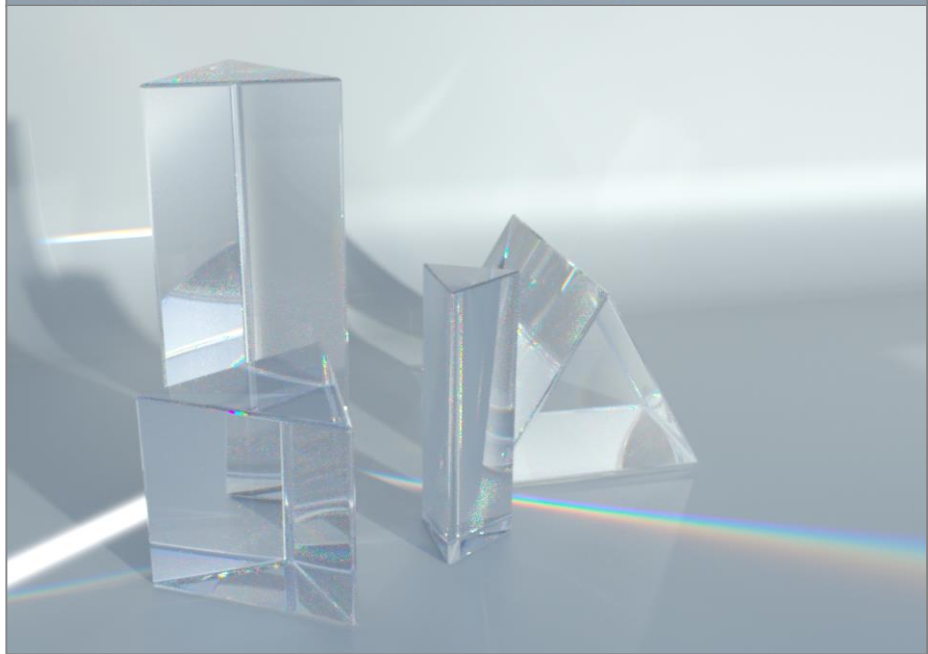


How Does it Work?

99% physically based Path Tracing

To guarantee simulation quality and ***Push Button***

- Limit shortcuts and good enough hacks to minimum
- Brute force (spectral) simulation
  - no intermediate filtering
  - scale over multiple GPUs and hosts even in interactive use
- Two-way path tracing from camera and (opt.) lights



How Does it Work?

99% physically based Path Tracing

To guarantee simulation quality and ***Push Button***

- Limit shortcuts and good enough hacks to minimum
- Brute force (spectral) simulation
  - no intermediate filtering
  - scale over multiple GPUs and hosts even in interactive use
- Two-way path tracing from camera and (opt.) lights
- Use NVIDIA Material Definition Language (MDL)

How Does it Work?

99% physically based Path Tracing

To guarantee simulation quality and ***Push Button***

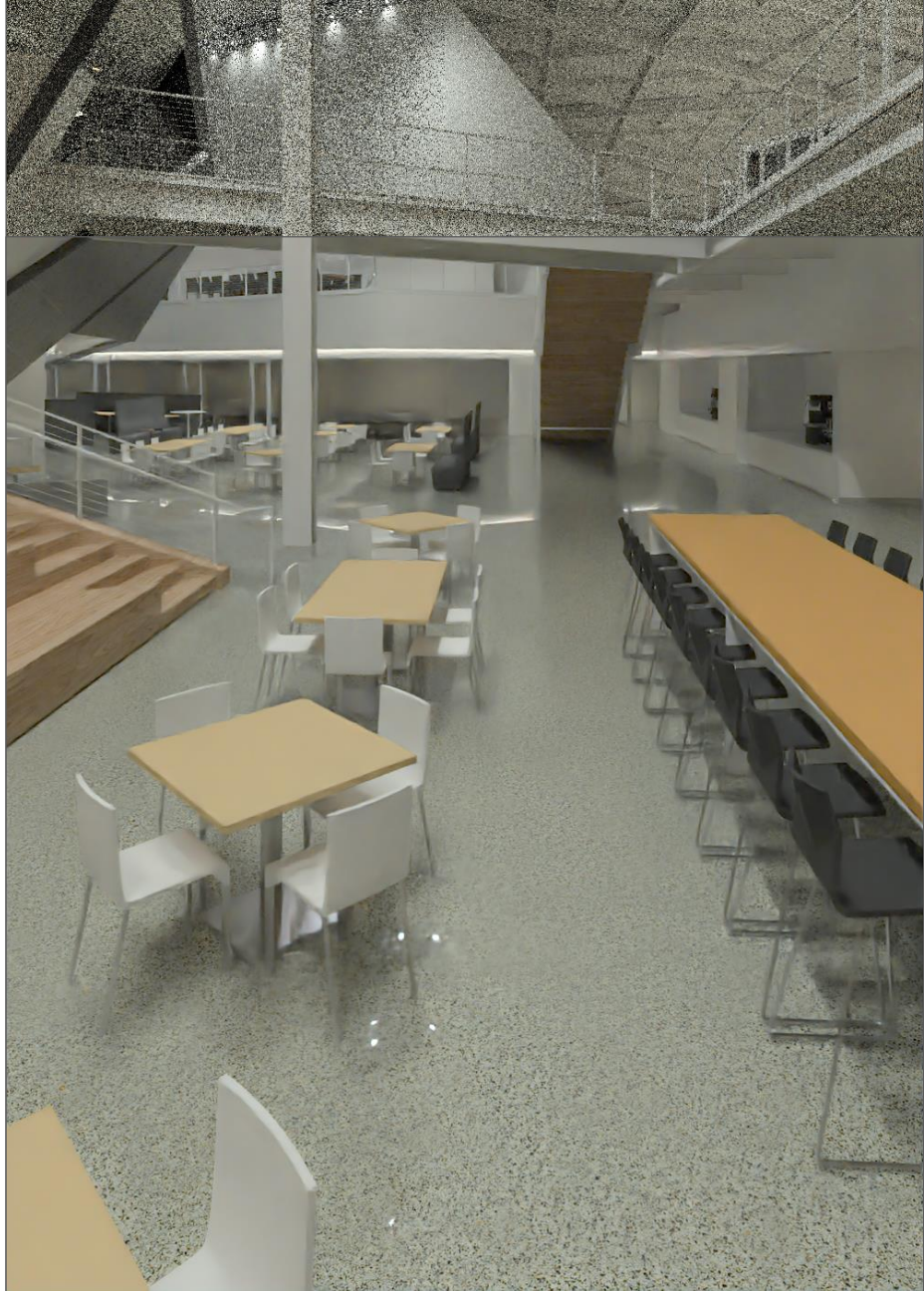
- Limit shortcuts and good enough hacks to minimum
- Brute force (spectral) simulation
  - no intermediate filtering
  - scale over multiple GPUs and hosts even in interactive use
- Two-way path tracing from camera and (opt.) lights
- Use NVIDIA Material Definition Language (MDL)
- NVIDIA AI Denoiser to clean up remaining noise

How Does it Work?

99% physically based Path Tracing

To guarantee simulation quality and ***Push Button***

- Limit shortcuts and good enough hacks to minimum
- Brute force (spectral) simulation
  - no intermediate filtering
  - scale over multiple GPUs and hosts even in interactive use
- Two-way path tracing from camera and (opt.) lights
- Use NVIDIA Material Definition Language (MDL)
- NVIDIA AI Denoiser to clean up remaining noise



Wavefront Architecture

From Megakernel

Follows each path to completion

One path at a time

Single CUDA (mega-)kernel

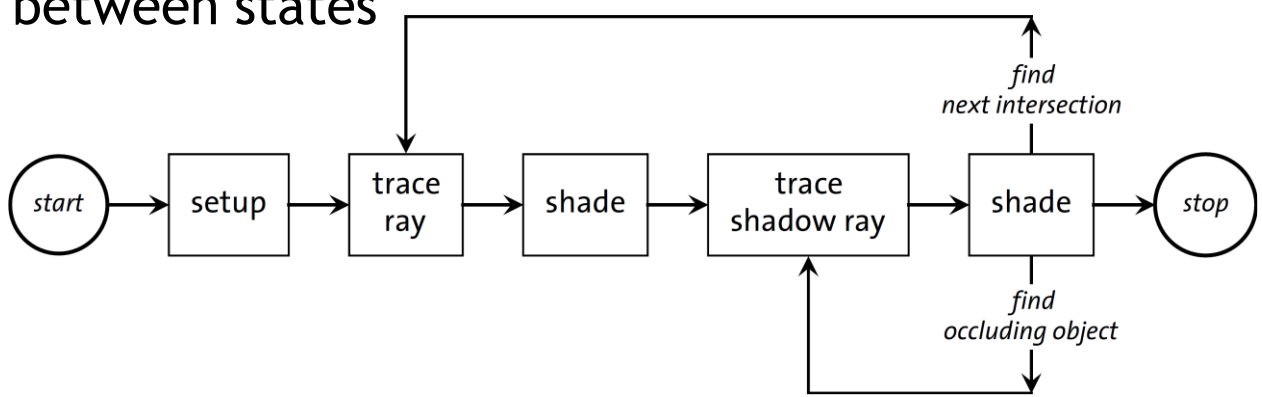
to State Machine

Small progress on each path per step

Millions of *active* paths at a time

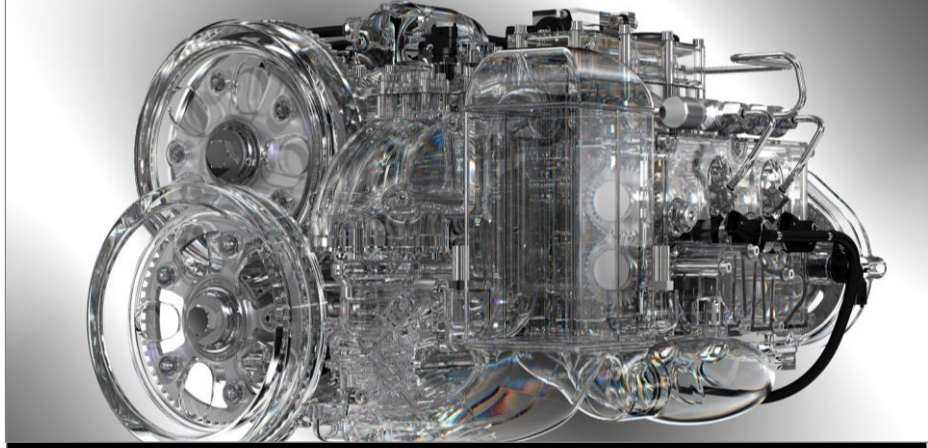
Multiple smaller CUDA kernels (states) specialized on parts of the simulation (state machine)

Global memory (*AoSoA* layout) to *communicate* between states



Wavefront Architecture

Iray State Machine



23 specialized CUDA kernels (scene dependent)

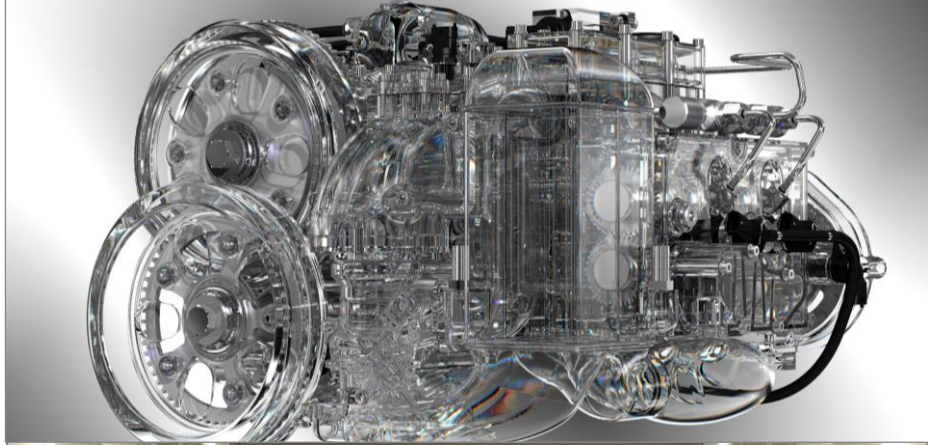
- Ray tracing
  to complete a path camera⬌light
  and connecting to lights on the way (NEE)

Wavefront Architecture

Iray State Machine

23 specialized CUDA kernels (scene dependent)

- Ray tracing
  to complete a path camera⬌light
  and connecting to lights on the way (NEE)
- Geometry / textured-light and environment importance sampling



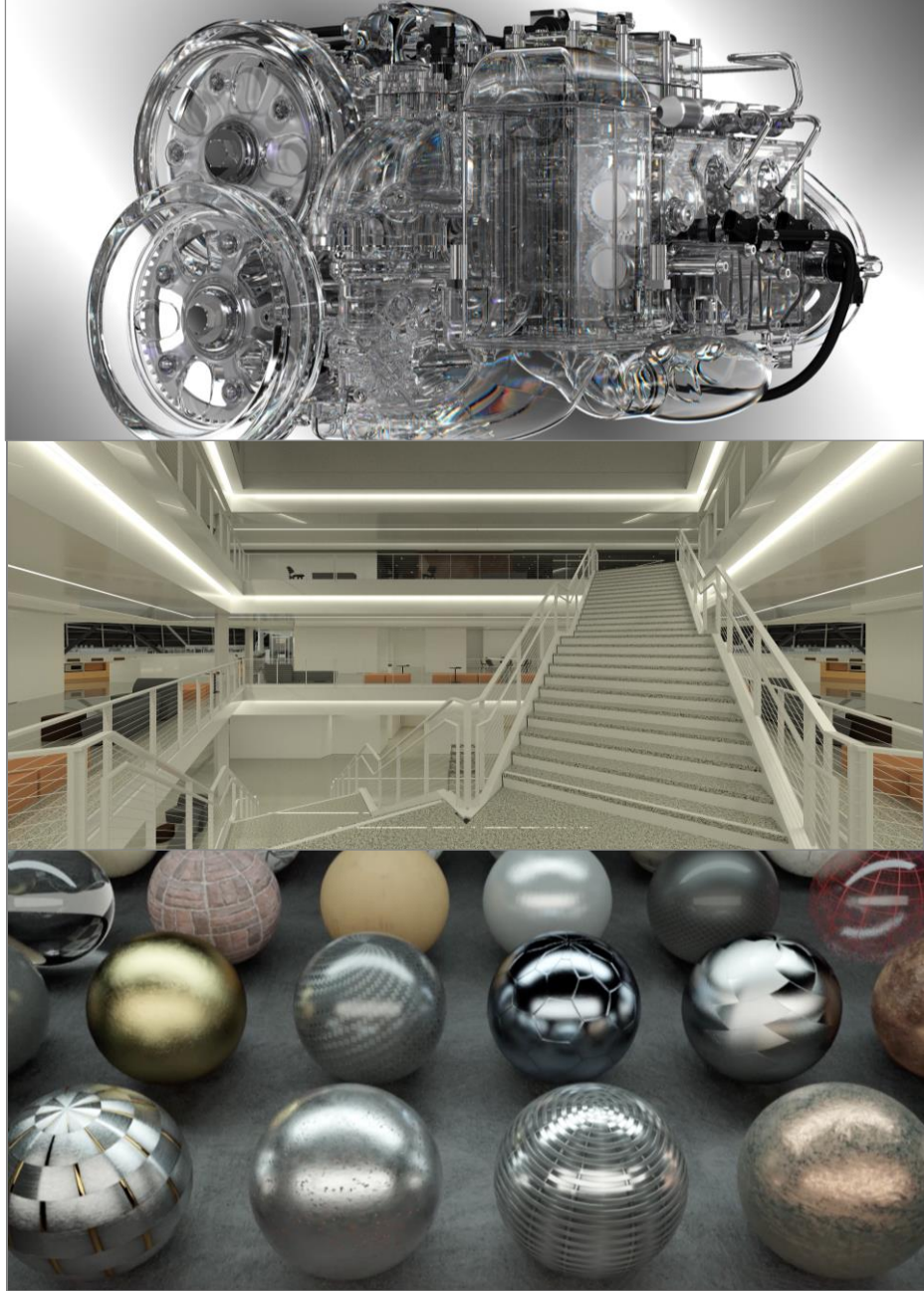
~400.000 emissive triangles

Wavefront Architecture

Iray State Machine

23 specialized CUDA kernels (scene dependent)

- Ray tracing
  to complete a path camera⬌light
  and connecting to lights on the way (NEE)
- Geometry / textured-light and environment importance sampling
- Material evaluation / importance sampling
- …



Wavefront Architecture

Iray State Machine

Tail-megakernel to finish up the last handful of paths

State machine within a single kernel to reduce kernel launches

Techreport: *The Iray Light Transport Simulation and Rendering System*
https://arxiv.org/pdf/1705.01263.pdf

Adding RTX Support

From OptiX Prime to OptiX 7

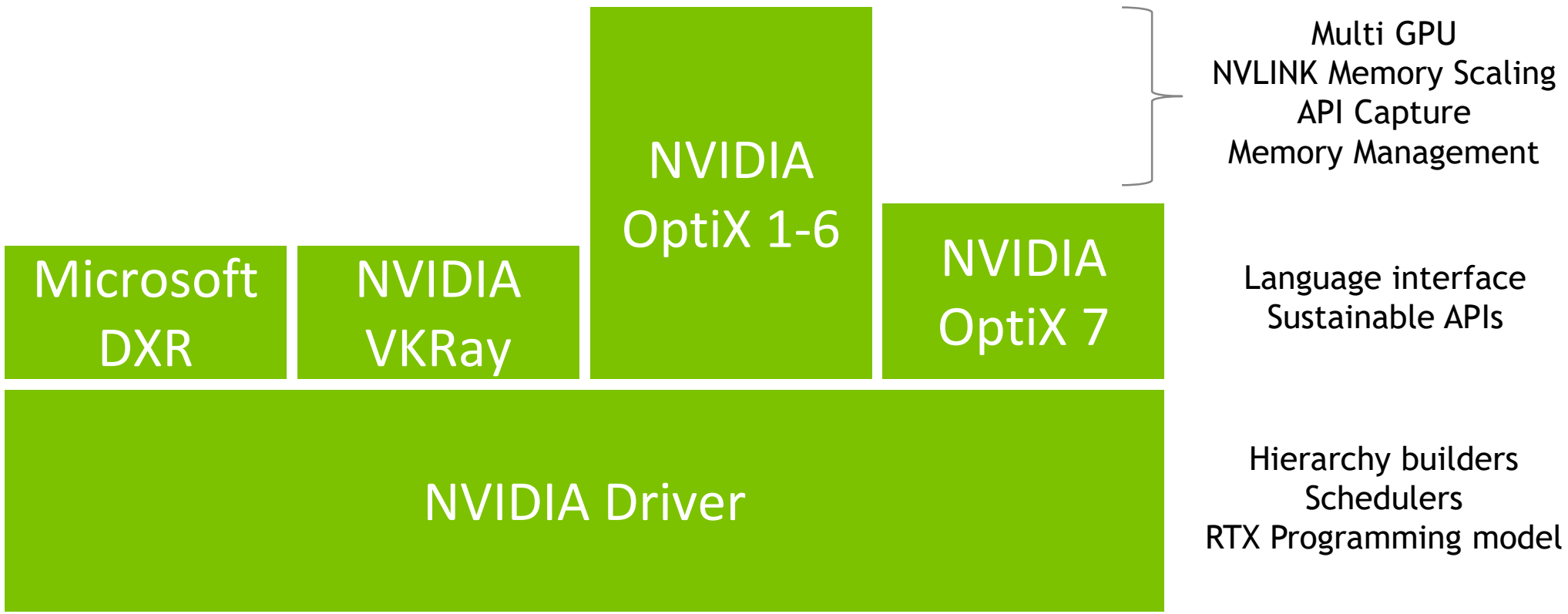
Dec 2018: Start with RTX prototype

Feb 2019: Start using WIP OptiX 7 implementation

May 2019: Shipping!

First Iray RTX image

Introducing OptiX 7

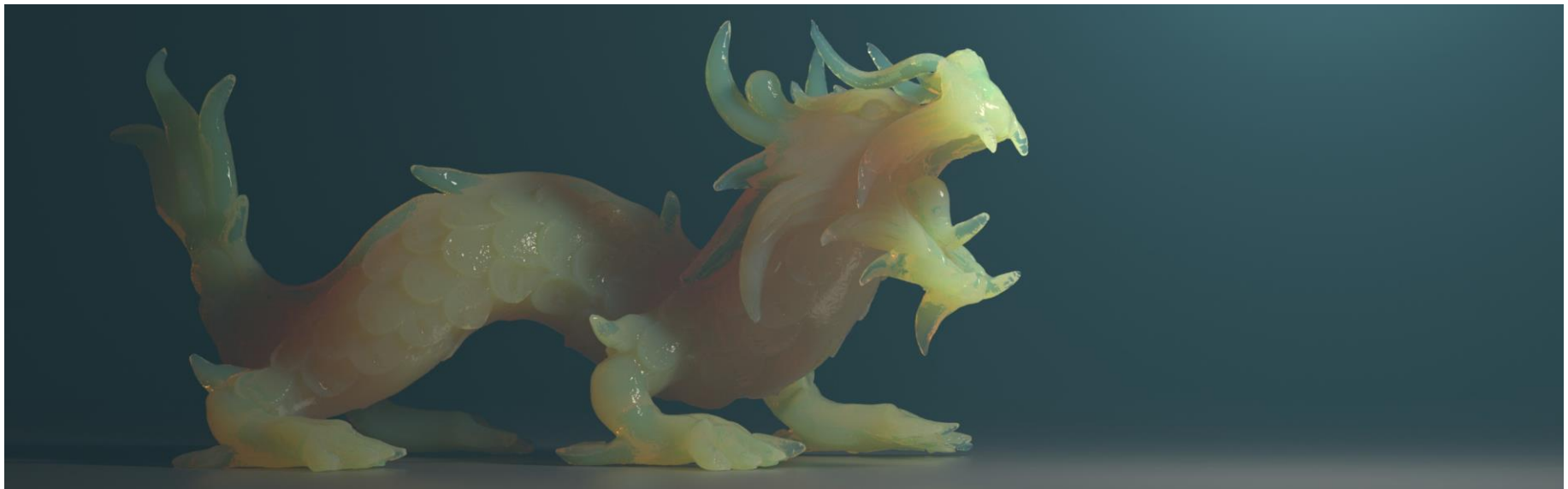


Iray on OptiX 7

Wavefront Architecture

All kernel variants that need to trace rays are now executed through OptiX 7

- Path-/Light-Tracer main trace kernels
  - incl. SSS code and shortcuts for state machine early outs



Iray on OptiX 7

Wavefront Architecture

All kernel variants that need to trace rays are now executed through OptiX 7

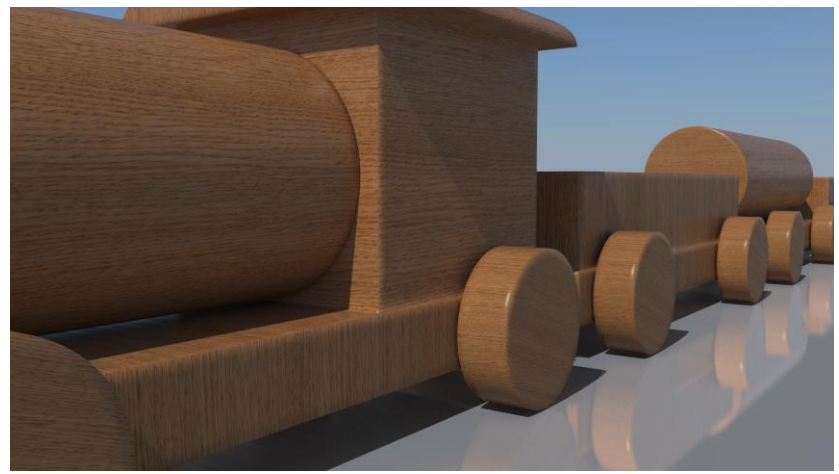
- Path-/Light-Tracer main trace kernels
  - incl. SSS code and shortcuts for state machine early outs
- Path-/Light-Tracer shadow trace kernels
  - incl. few shortcuts for state machine early outs

Iray on OptiX 7

Wavefront Architecture

All kernel variants that need to trace rays are now executed through OptiX 7

- Path-/Light-Tracer main trace kernels
  - incl. SSS code and shortcuts for state machine early outs
- Path-/Light-Tracer shadow trace kernels
  - incl. few shortcuts for state machine early outs
- Rounded Corners



Iray on OptiX 7

Wavefront Architecture

All kernel variants that need to trace rays are now executed through OptiX 7

- Path-/Light-Tracer main trace kernels
  - incl. SSS code and shortcuts for state machine early outs
- Path-/Light-Tracer shadow trace kernels
  - incl. few shortcuts for state machine early outs
- Rounded Corners
- Light-Tracer lens connection

Iray on OptiX 7

Wavefront Architecture

All kernel variants that need to trace rays are now executed through OptiX 7

- Path-/Light-Tracer main trace kernels
  - incl. SSS code and shortcuts for state machine early outs
- Path-/Light-Tracer shadow trace kernels
  - incl. few shortcuts for state machine early outs
- Rounded Corners
- Light-Tracer lens connection

All other kernels stay on plain CUDA implementations / kernel launches

Iray on OptiX 7

Wavefront Architecture

Split up the Tail-megakernel into 2 new kernels
  Trace rays + the *remainder* of the state machine

Majority of code in `__raygen__`
  One single `optixTrace()` call, no branching, for best performance
  (except for Tail-trace- and rounded corners kernels)

`__closesthit__` directly fills wavefront state, no payload communication

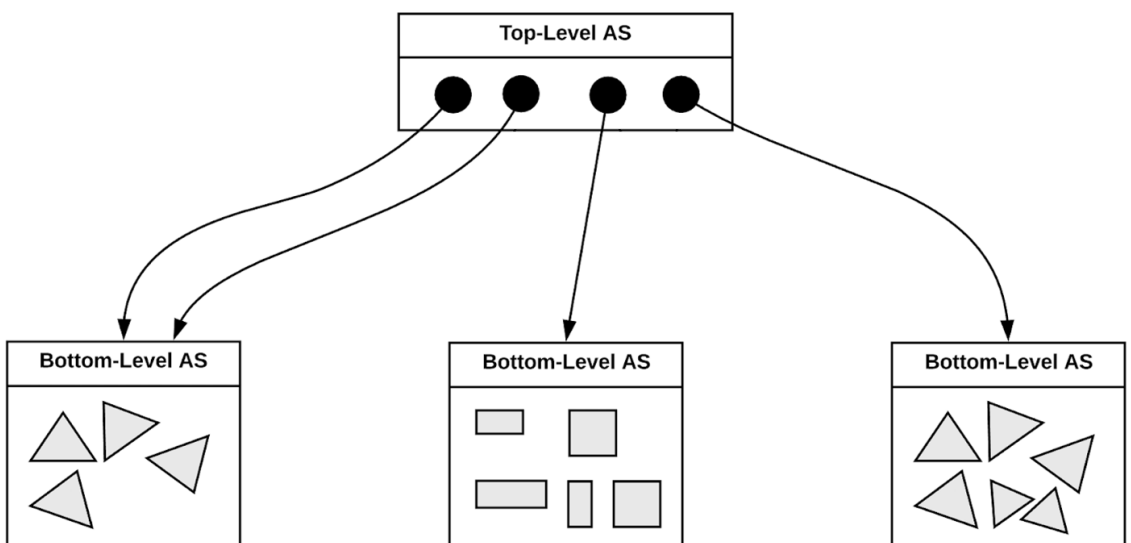
Compile time / Pipeline setup 7-10 secs (with warm cache 0.1-0.2 secs)

~21k lines of PTX

Iray on OptiX 7

RTX Hierarchy Setup

2-level hierarchy to get full RT core performance
  optionally: reduce instancing overhead by (partially) flattening instances



Iray on OptiX 7

RTX Hierarchy Setup

2-level hierarchy to get full RT core performance
  optionally: reduce instancing overhead by (partially) flattening instances

Use compaction
  slight build time decrease not that much of an issue for us
  memory savings can be dramatic

No native OptiX 7 Motion Blur to get full RT cores performance
  as sample rate per pixel is high and hierarchy updates cheap,
  do brute force sample trafos/materials and rebuild scene every X iterations

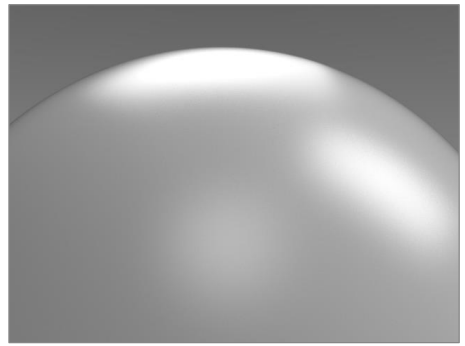
Refitting of bottom level hierarchies for vertex deformed geometry

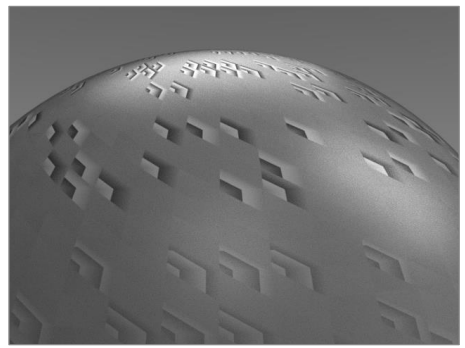
General Issues

Precision / Performance / Memory Usage

4 Ray Tracing implementations at work now (before: OptiX Prime)

- Embree (CPU)
- OptiX Prime (pre-Turing / need to support all CUDA 10 GPUs)
- OptiX 7 / RT Cores (Turing)
- OptiX 7 / RTX Software Traversal (Turing with no RT Cores)





Slightly different behavior in special cases (i.e. self intersection) and hierarchy construction/data implementation details

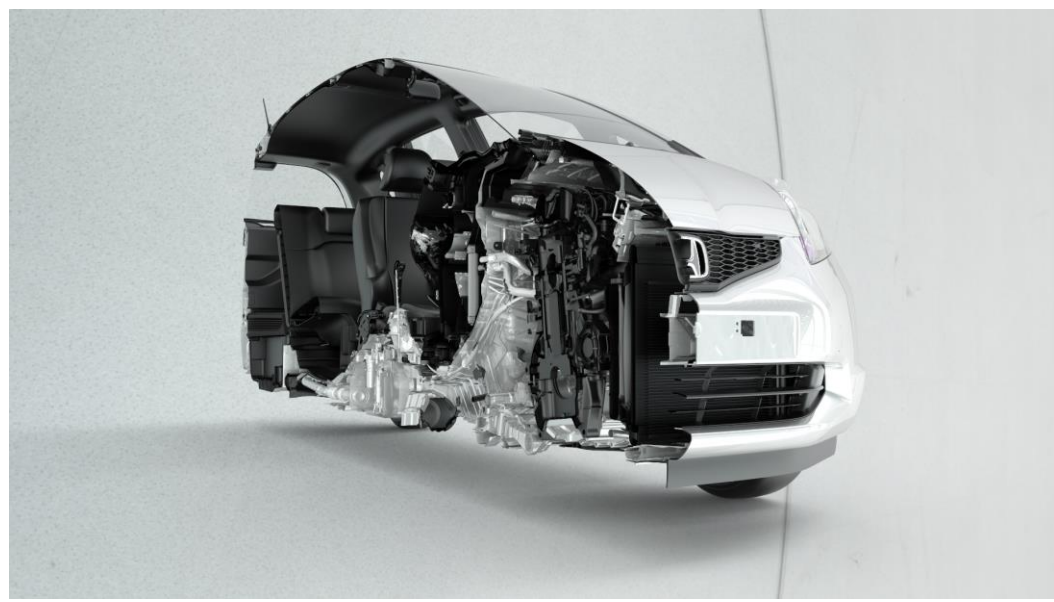
General Issues

Performance

Triangle/Node intersection watertightness has some interesting implications
- Origins very far away with directions pointing to the scene will intersect almost the whole scene, causing massive slowdowns

- Iray: infinite ground plane / shadow catcher generates this frequently

- Workaround by manually pushing all ray origins closer to the scene BBox



Performance

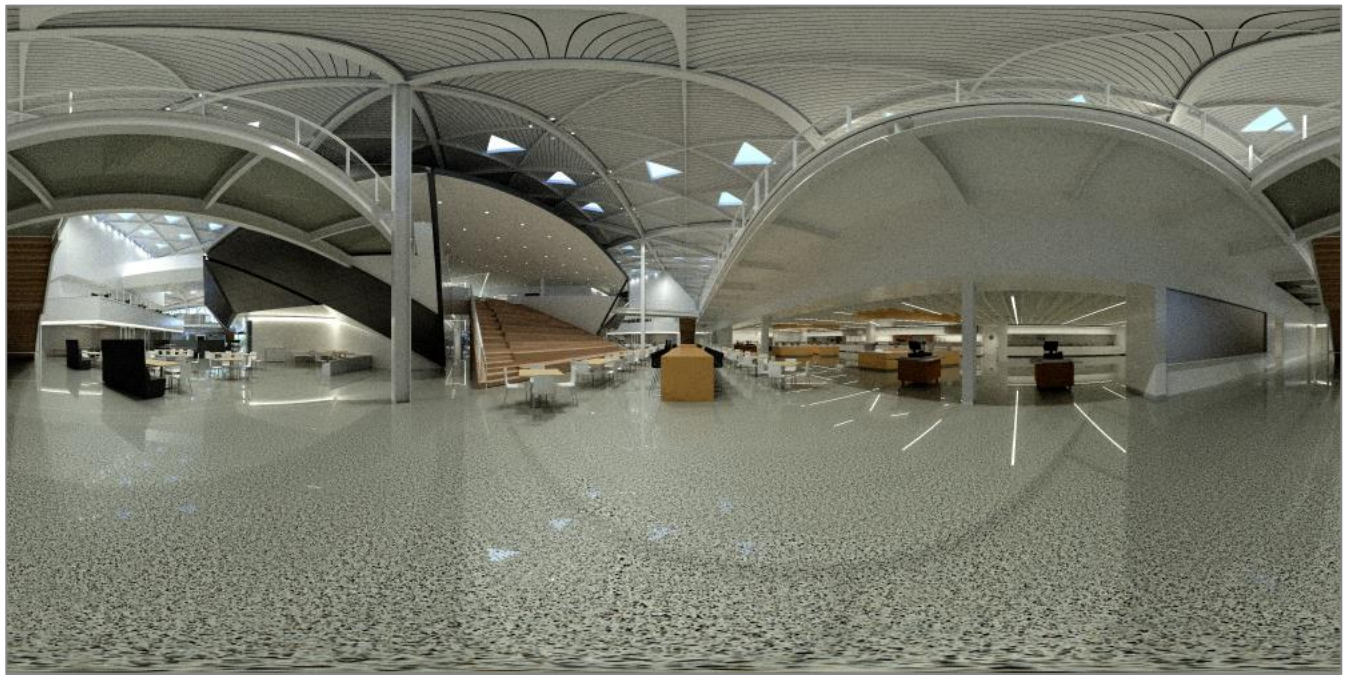
3.0 x Overall Rendering Speed-Up

Performance

2.8 x Overall Rendering Speed-Up

Performance

1.5 x Overall Rendering Speed-Up



~101k instances flattened to ~50m triangles
66s down to 61s (4k, RTX 6000)

Performance

1.05 x Overall Rendering Speed-Up

Customers see full range from “no” to 5x overall rendering speed-up

Ray Tracing no Bottleneck Anymore

When Using RT Cores

One still has to care about *overall efficient* rendering:
Not just tracing a ray as fast as possible, but generating *valuable rays* / samples

- Sample and eval large, layered material and texture node graphs
- Sample and eval large amount of geometric light sources

i.e. many instructions and a lot of memory accessed per traced ray

Otherwise many many more rays/paths needed to get similar noise level
Need to balance generation time vs sample quality vs evaluation time vs trace time

Performance

Notes

Batch scheduling (e.g. long running renderings / cloud)
efficient with current Iray OptiX 7 implementation
triggers almost no Tail-megakernel (paths are regenerated on the fly)

Interactive scheduling suffers from split of
Tail-megakernel: kernel launch overhead too high

Too much time spent in light importance sampling
traversal of geometry lights & environment light
hierarchies

Going Forward

RTX Specific Roadmap

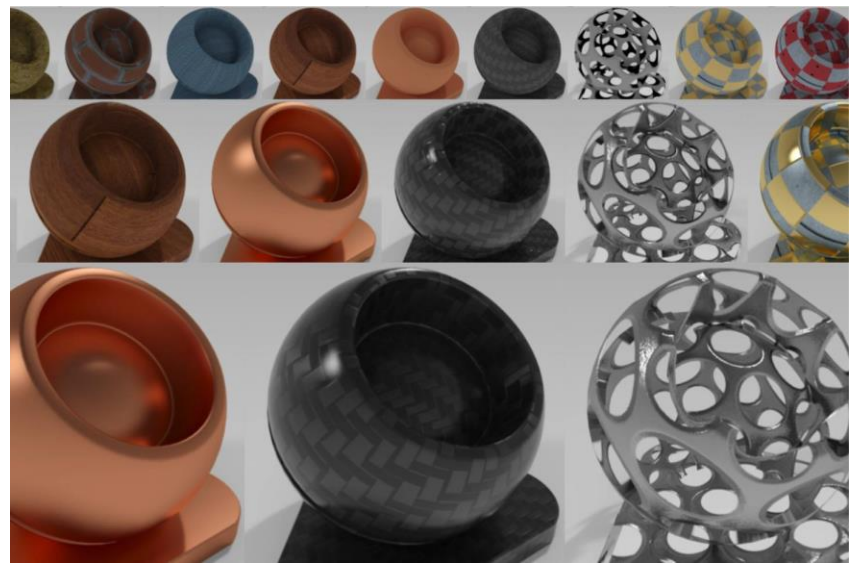
Optimize / Rethink importance sampling and material / tex evaluation pipelines to shift work-per-sample-ratio towards Ray Tracing again

Reduce material / texture complexity dynamically (LOD via MDL distiller)

Adaptive Sampling

Over time: Better scheduling performance / less overhead by basing complete core on OptiX 7

…



Going Forward

RTX Specific Roadmap

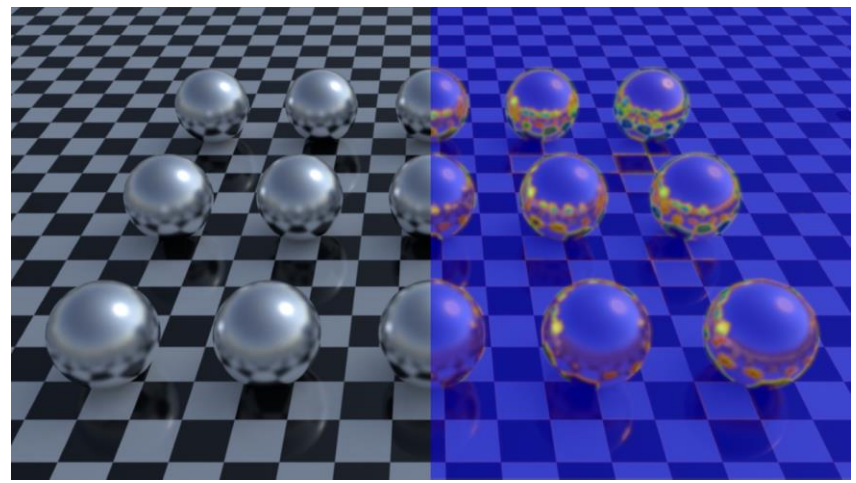
Optimize / Rethink importance sampling and material / tex evaluation pipelines to shift work-per-sample-ratio towards Ray Tracing again

Reduce material / texture complexity dynamically (LOD via MDL distiller)

Adaptive Sampling

Over time: Better scheduling performance / less overhead by basing complete core on OptiX 7

…



Questions?

Acknowledgments

Carsten Wächter
Daniel Seibert
Enzo Catalano
Matthias Raab

More Information

Techreport: *The Iray Light Transport Simulation and Rendering System*
https://arxiv.org/pdf/1705.01263.pdf