Heterogeneous Parallel Computing for Rendering Large-Scale Data

Sung-eui Yoon
Associate Professor
KAIST

http://sglab.kaist.ac.kr

Acknowledgements

- Collaborators
  - My students, M. Gopi, Miguel Otaduy, George Drettakis, SeungYoung Lee, YuWing Tai, John Kim, Dinesh Manocha, Peter Lindstrom, Yong Joon Lee, Pierre-Yves Laffont, Jeong Mo Hong, Sun Xin, Nathan Carr, Zhe Lin

- Funding sources
  - Boeing, Adobe, Samsung
  - AMD, Microsoft Research Asia
  - Korea Research Foundation
  - MSIP, IITP

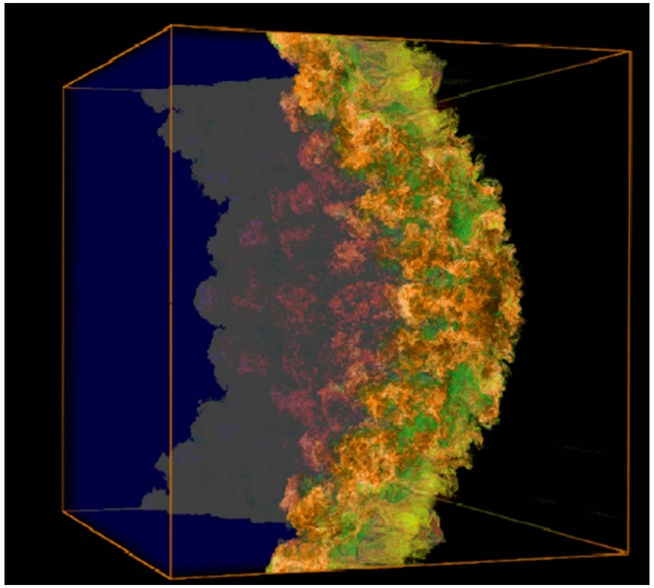
KAIST

Past: Rendering Massive Geometric Data

Boeing 777, 470 M tri.

Large-scale virtual world, 83 M tri.

Scanned model, 372 M tri. (10 GB)



Over 3 Terabytes of geometric data

Present: Scalable Ray Tracing, Image Search, Motion Planning

- **Designing *scalable graphics and geometric algorithms* to efficiently handle massive models on commodity hardware**

Photo-realistic rendering

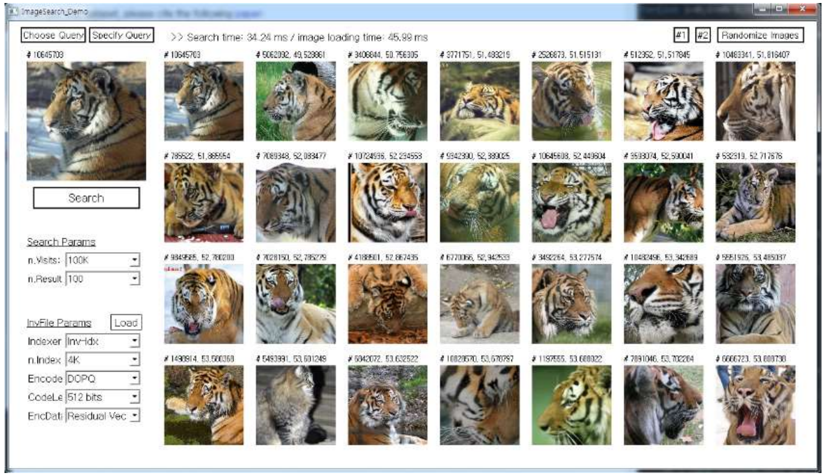


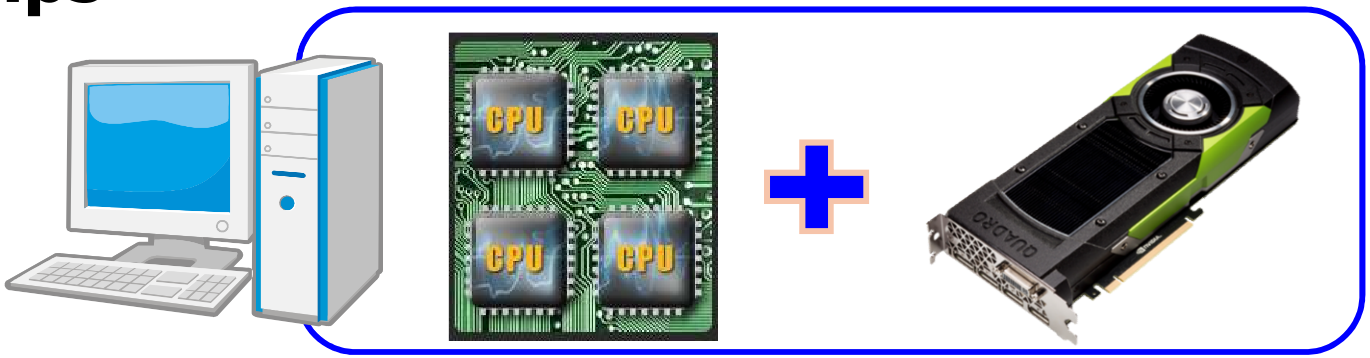
Image search

Motion planning

KAIST

Recent Hardware Trends

- Multi and many cores
  - CPUs and GPUs are increasing the # of cores
- Heterogeneous architectures
  - Intel Sandy Bridge, AMD Fusion, and Nvidia Tegra embedded chips

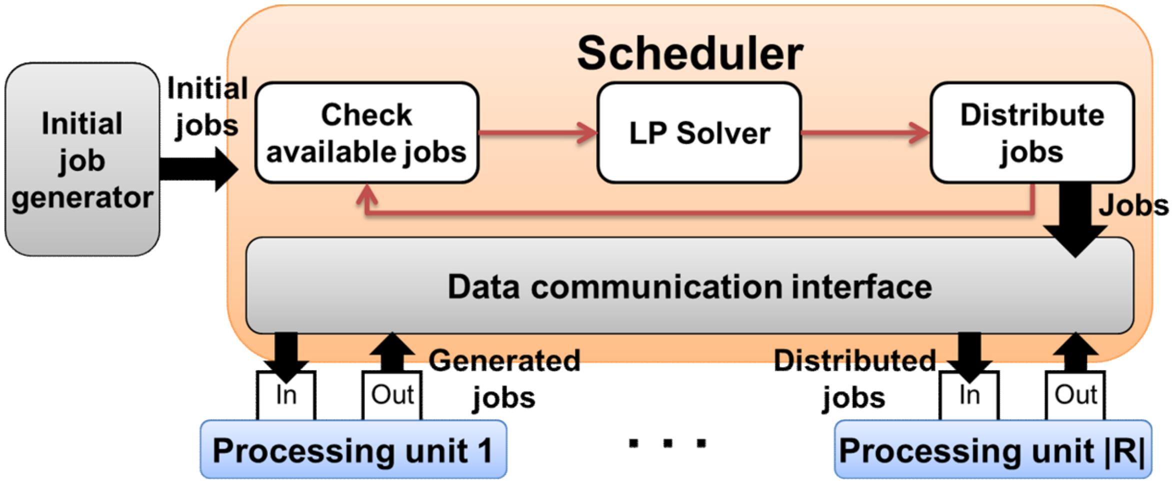


Images from NVIDIA

- Previous approaches
  - Utilize either multi-core CPUs or GPUs

Hybrid Parallel Computation for Proximity Queries

- **Our initial work: manually assign jobs of continuous collision detection to CPUs and GPUs**
  - **Received a best paper award at Pacific Graphics, 09**
- **A general, job distribution algorithm for CPUs and GPUs [Kim et al., TVCG 13, Spotlight paper]**

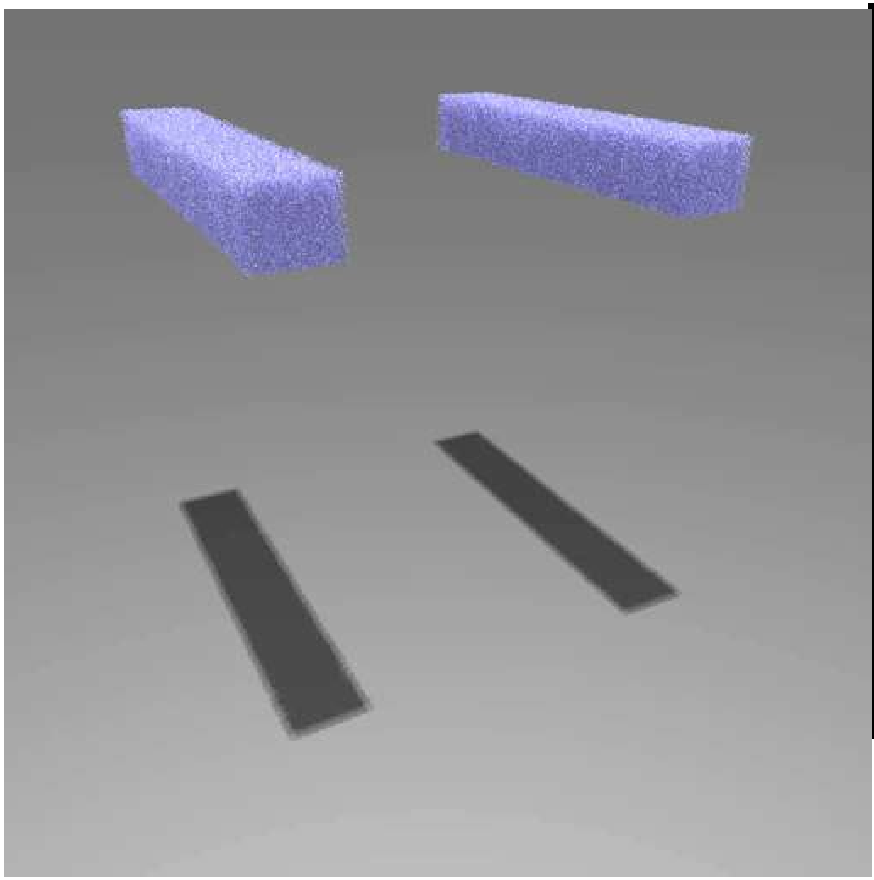


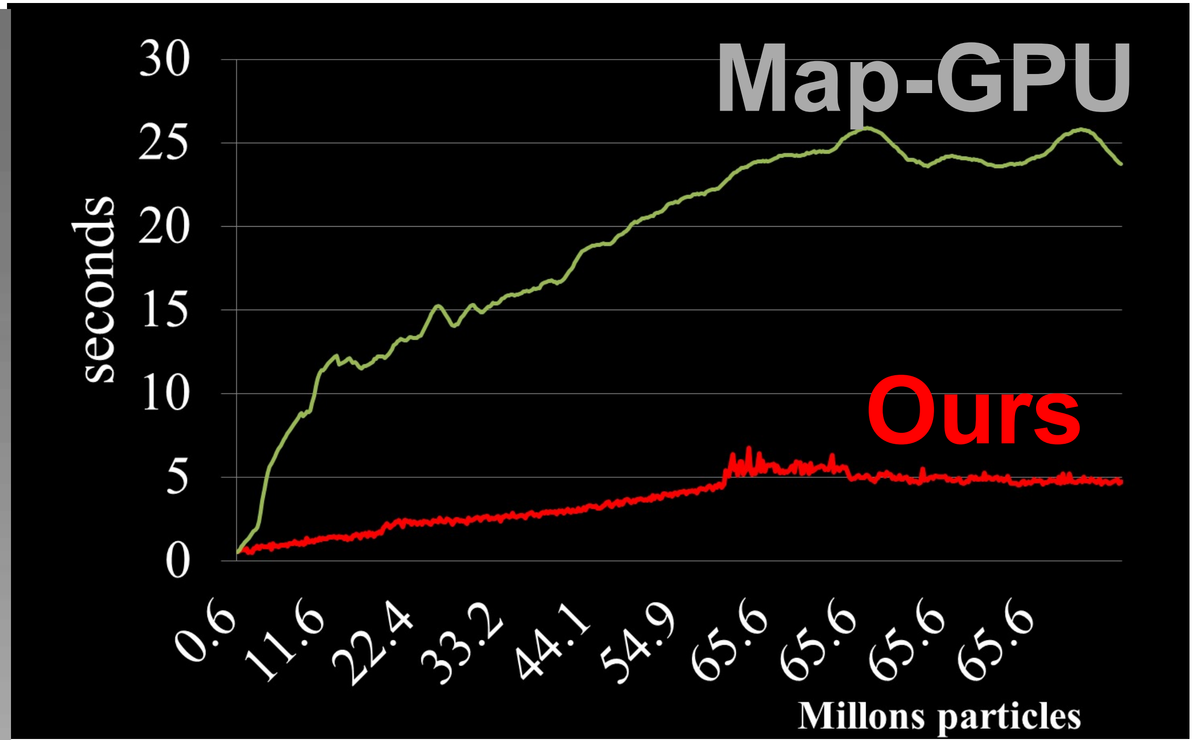
Motion planning

[Lee et al., ICRA 12]

Out-of-Core Proximity Computation for Particle-based Fluid Simulations [Kim et al., HPG 14]

Two hexa-core CPUs w/ 192 GB RAM
GeForce GTX 780) with 3 GB video RAM





Up to 65.6 M Particles
Maximum data size: 13 GB

NVIDIA mapped memory Tech
- Map CPU memory space
  into GPU memory address space

KAIST

Heterogeneous Parallel Computing for Rendering

- **T-ReX: Interactive Global Illumination of Massive Models on Heterogeneous Computing Resources, IEEE TVCG 2014**
  - **Manually assign tasks to CPUs and GPUs**
  - **Source codes are available**

- **Timeline Scheduling for Out-of-Core Ray Batching, High Performance Graphics (HPG), 2017**
  - **Automatic task assignment for high performance**

T-ReX: Interactive Global Illumination of Massive Models on Heterogeneous Computing Resources

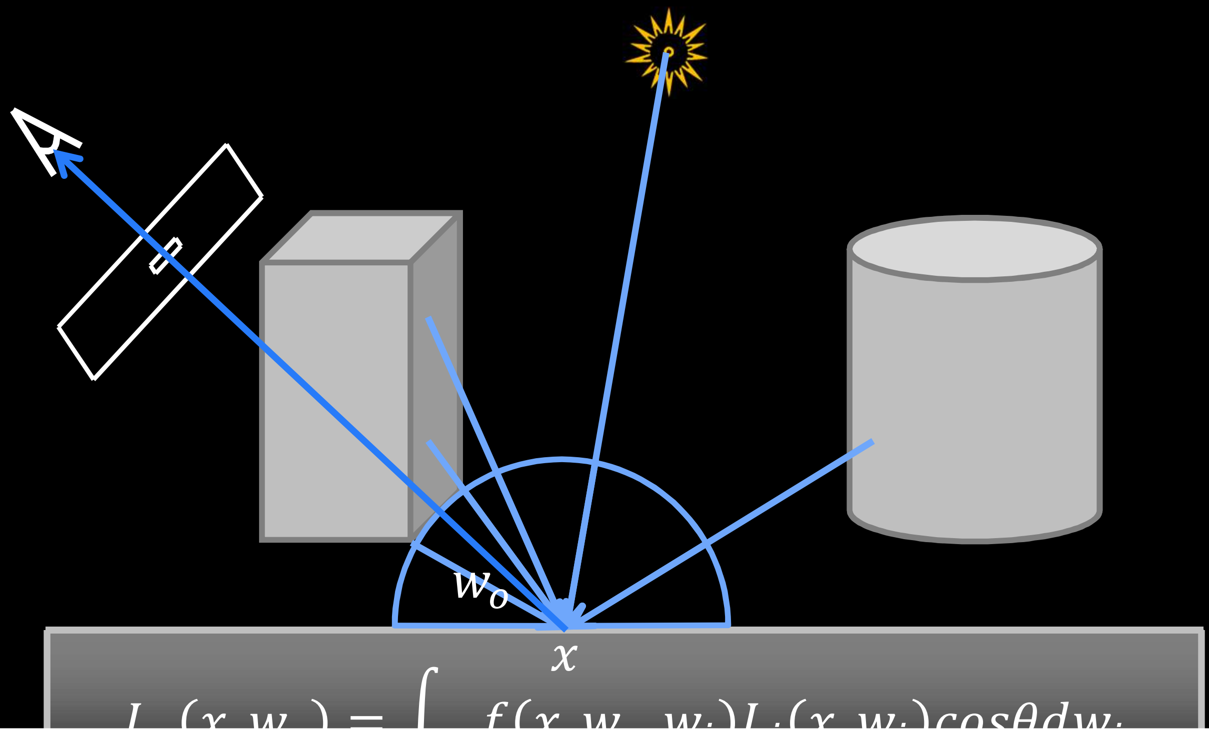
Tae-Joon Kim*, Xin Sun§, and Sung-Eui Yoon*

KAIST*, Microsoft Research Asia§

IEEE Transactions on Visualization and Computer Graphics (TVCG), 2014

Project Homepage with Codes:
http://sglab.kaist.ac.kr/T-ReX

Global Illumination



Enormous computation is necessary

Interactive Global Illumination

- Utilize GPU
- Use sparse voxel octrees
- Model complexity < 10 M tris.

Interactive Indirect Illumination Using Voxel Cone Tracing [Crassin et al., PG11]

Massive Models

- Due to advances of modeling, simulation, and data capture techniques

CAD oil tanker, 82 M tri. (4 GB)

Boeing 777, 366 M tri. (20 GB)

Scanned model, 372 M tri. (10 GB)

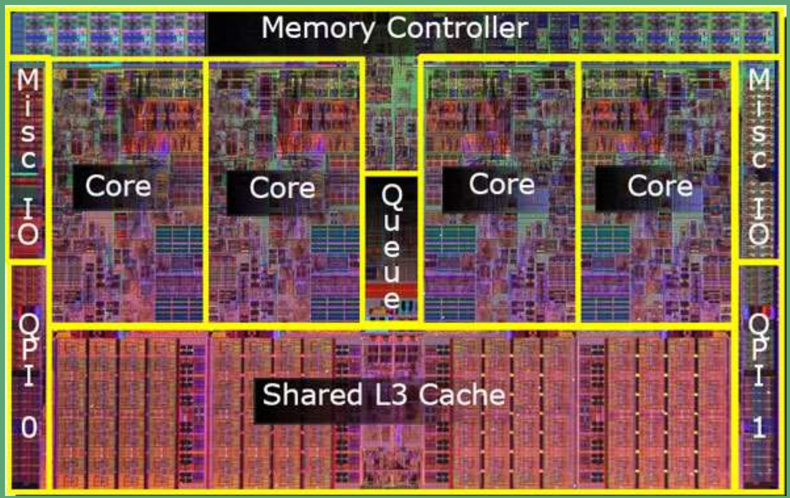
- Long data access time and low I/O performance

Motivation

- Global illumination of small models can be done interactively
  - Thanks to advance of GPU architecture
- Interactive global illumination with massive models is still challenging
  - Maximize computation throughput
  - Minimize I/O requirement

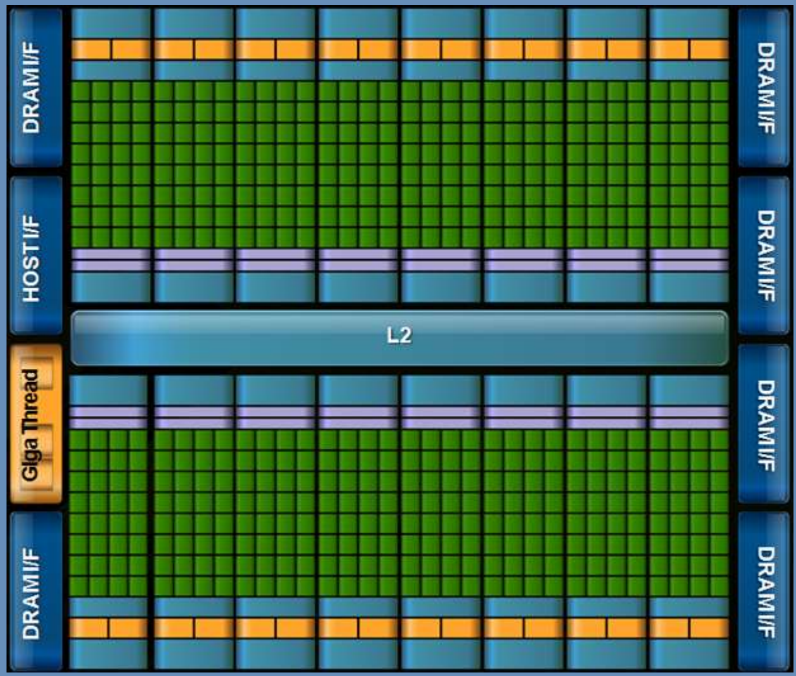
Heterogeneous Computing Resources

CPU



4 ~ 200 GB memory

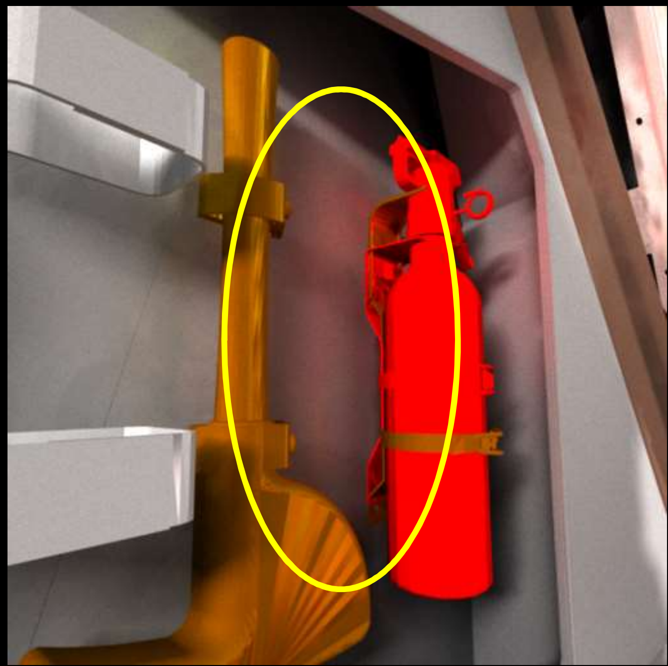
GPU



2 ~ 8 GB memory

Observation

- Global illumination effect is less sensitive to geometry details



Our Approach

- Hybrid approach

CPU

Geometric representation
(full detailed, large)

Compute direct illumination

Transmit intersection info.

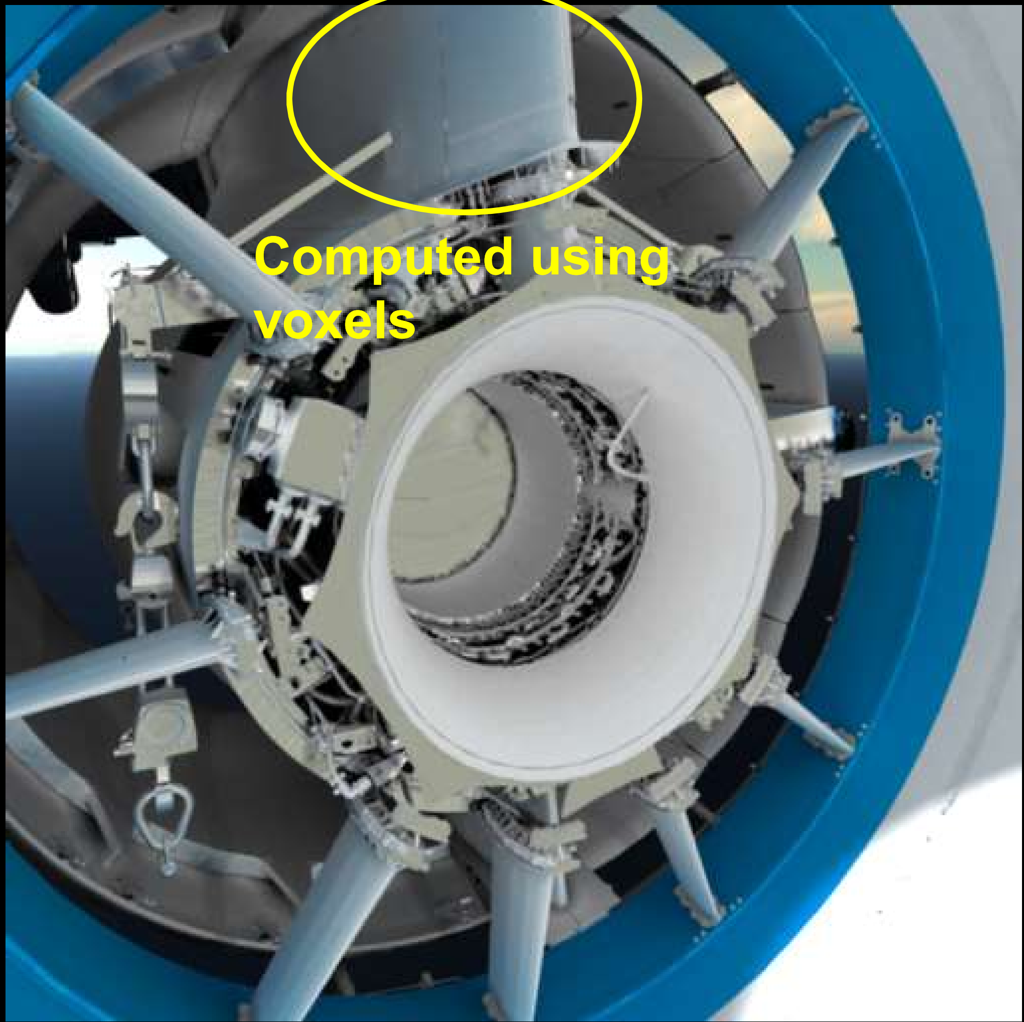
No mesh data trans.

GPU

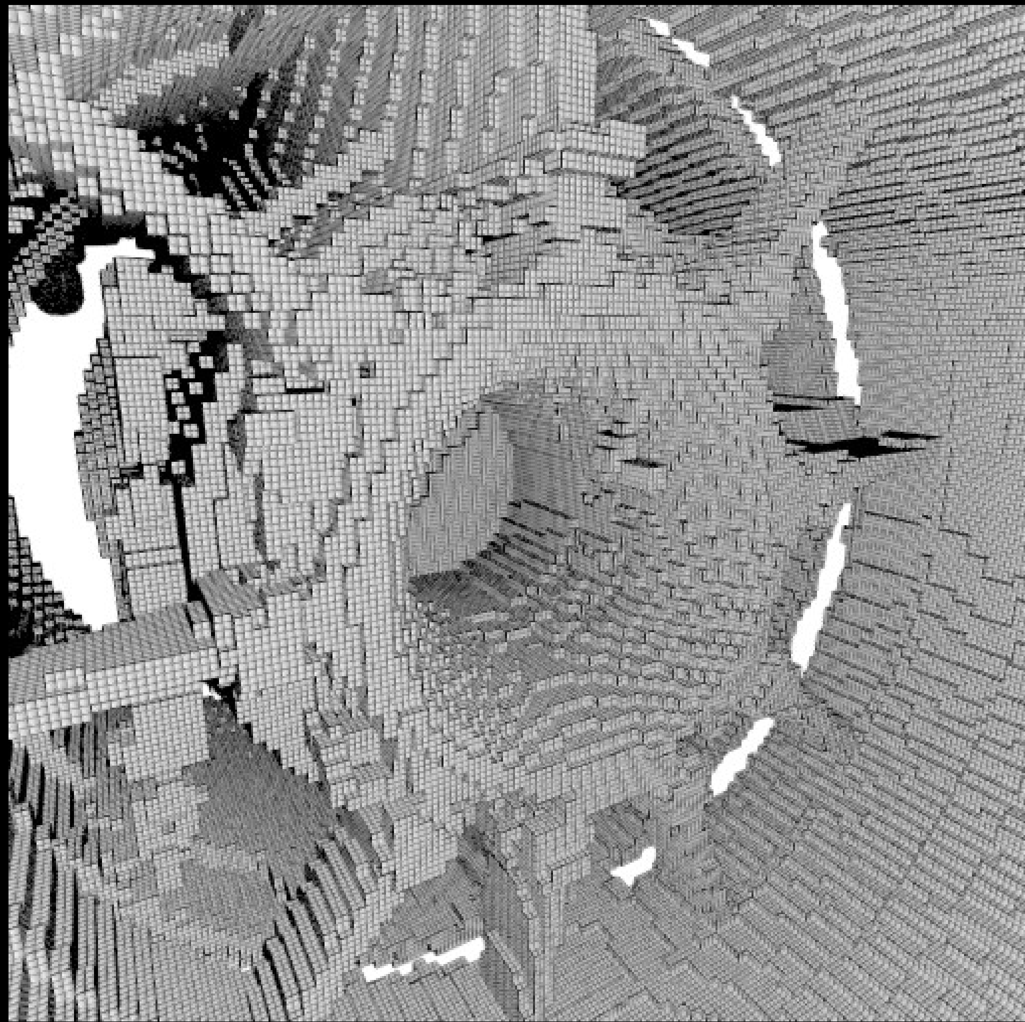
Volumetric representation of sparse voxel octree
(approximated, small)

Compute indirect illumination

Approximated Illumination



Raw Mesh visualization / After shading



Approximated voxel representation

Results

Outline

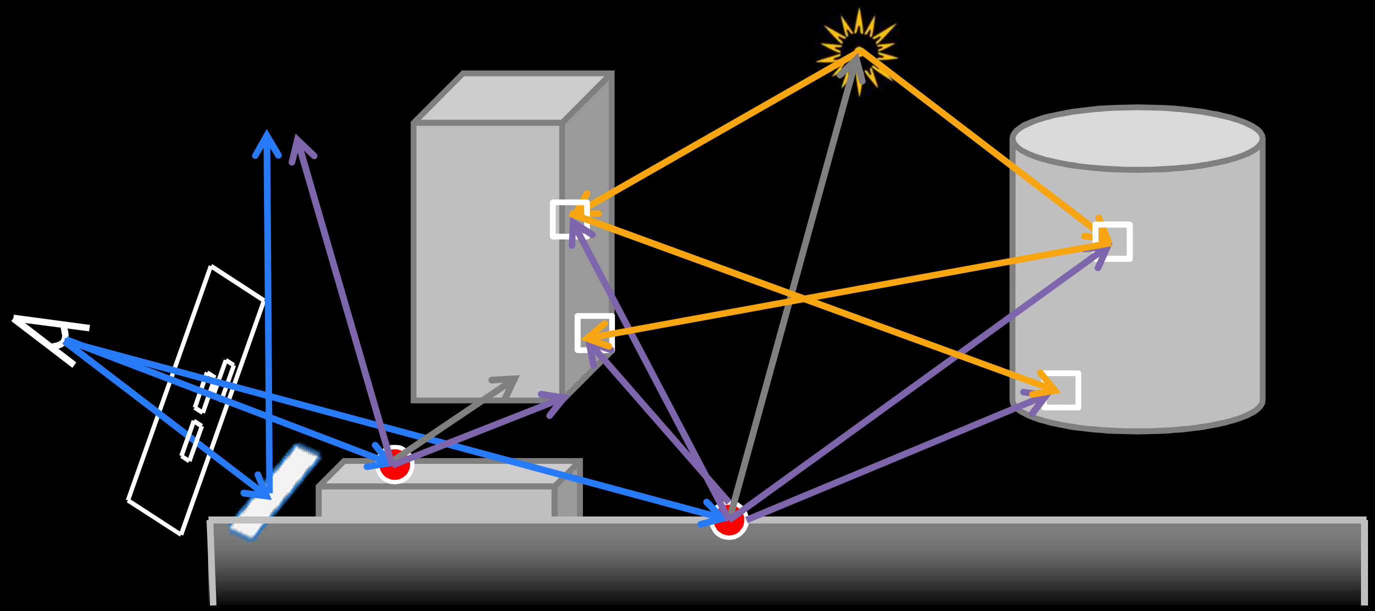
- Use photon mapping for rich visual effects

  e.g., color bleeding

- Classify rays into fitting processors
  - Each class of ray uses representation

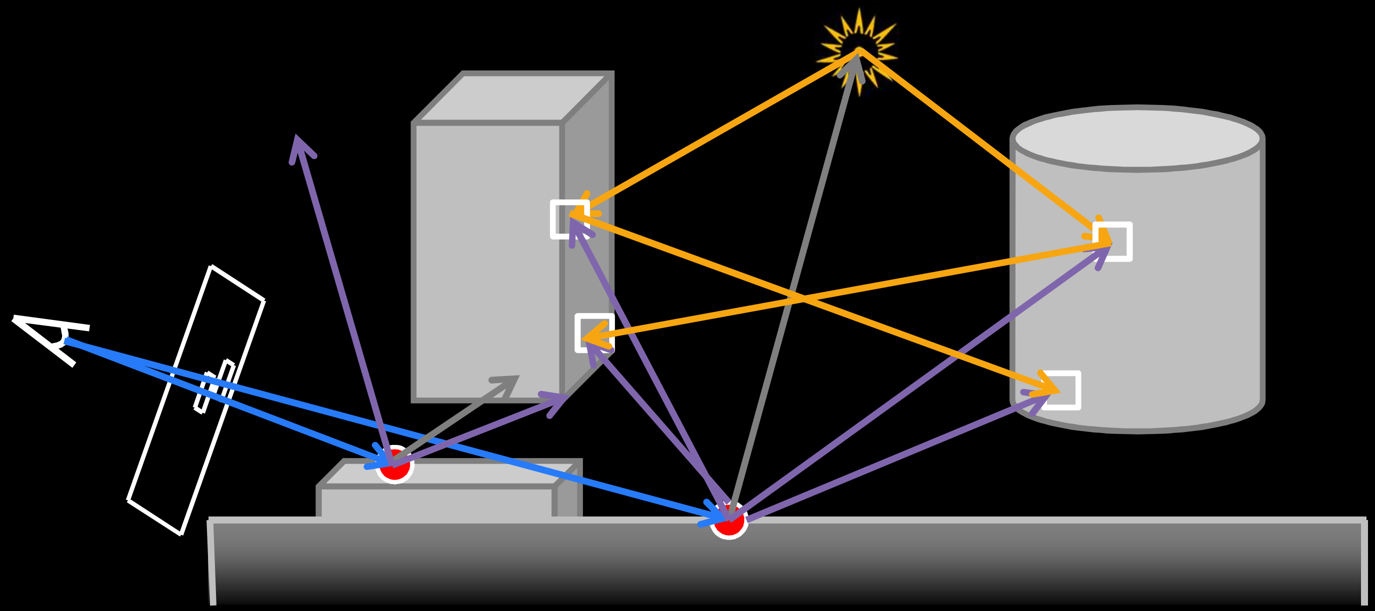
Ray Classification

- C-rays
  - More sensitive to geometry details
  - Generates high-frequency visual effects
  - The primary rays and their secondary rays reflected on perfect specular materials



Ray Classification

- G-rays
  - Less sensitive to geometry details
  - Generates low-frequency visual effects
  - Any rays other than C-rays (e.g., gathering rays, shadow rays)



Data Representations

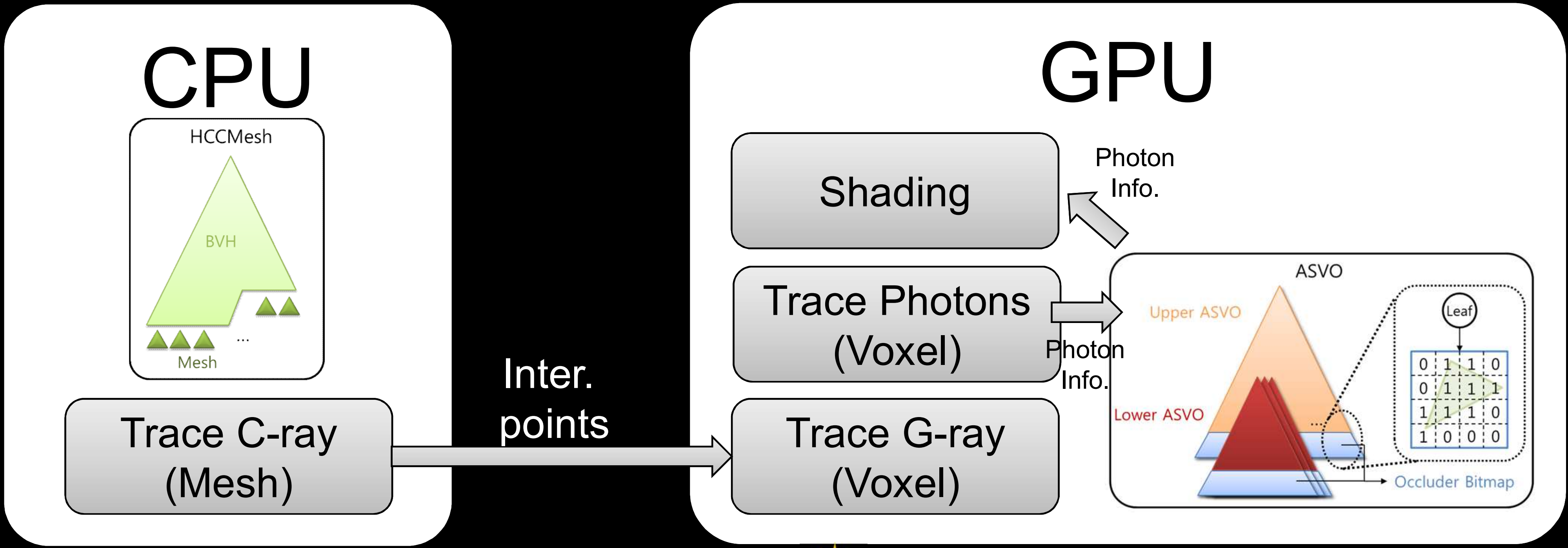
Augmented Sparse Voxel Octree (ASVO)

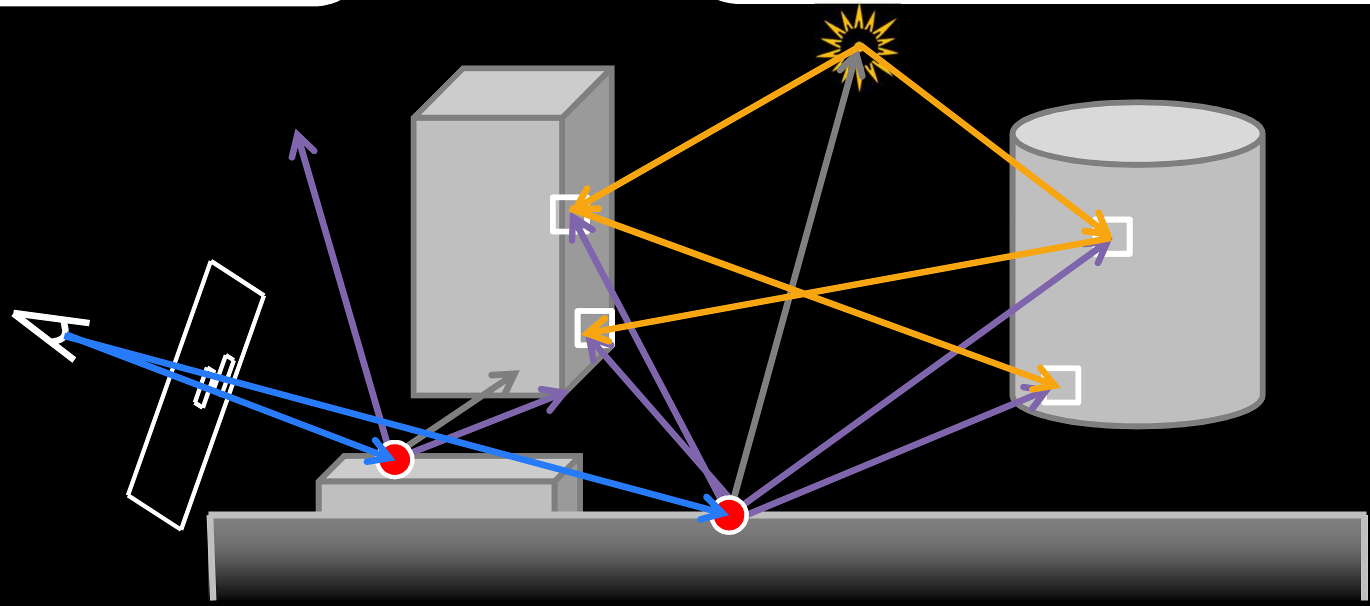
- GPU side volumetric representation for G-ray
- Efficiently traversed in GPU
- Approximated geometry & photon map

HCCMeshes [Kim et al. Eurographics'10]

- High quality geometry for C-ray
- Random-accessible compression (7:1 ~ 20:1)
- Supports high performance decompression

Rendering Process





Results

- Interactive responsiveness
  - About 30 ms response time for dynamic changes on cameras, materials, and lights
- High performance
  - 3 M ~ 20 M rays/s
- High complexity
  - Up to 470 M triangles

Results

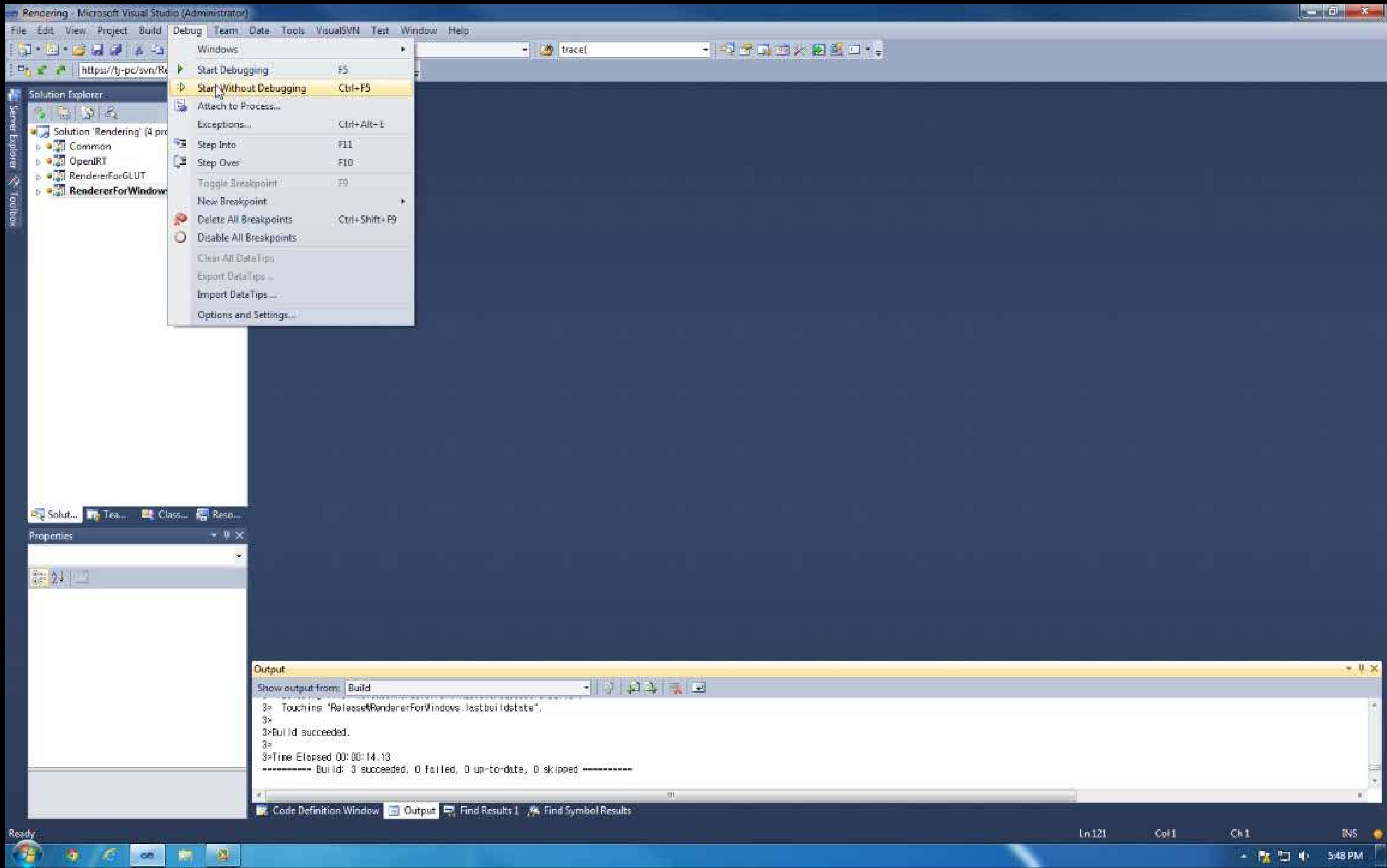
- Test environment (PC)
  - Intel Core i7 CPU (hexa-core) w/ 8 GB RAM
  - NVIDIA GTX 680 card with 2 GB DRAM
    - 15% of GPU memory was allocated for upper ASVO

- Boeing 777 model benchmark
  - 366M Triangles
  - 15.6 GB mesh + 21.8 GB BVH for raw model
  - 6.55 GB for HCCMesh
  - 11 area lights (generated 5 M photons each)

Comparison

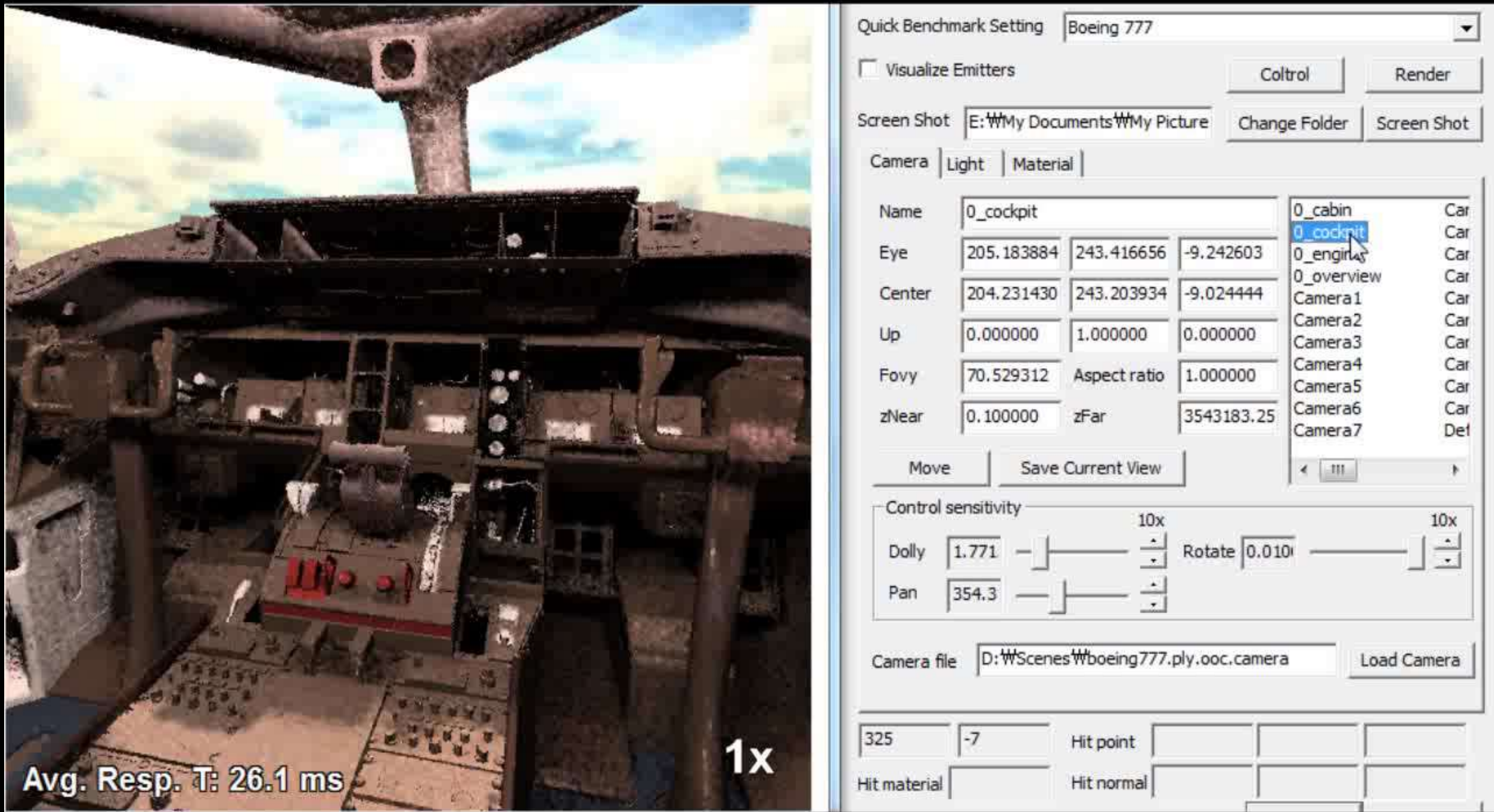
- 3.9 times improvement over CPU-only implementation
  - Same algorithm, but running on CPU only
  - Main memory holds both representations (HCCMeshes, ASVOs)
- 135 times improvement over simple photon mapping on CPU
  - Using HCCMeshes only

Demonstration

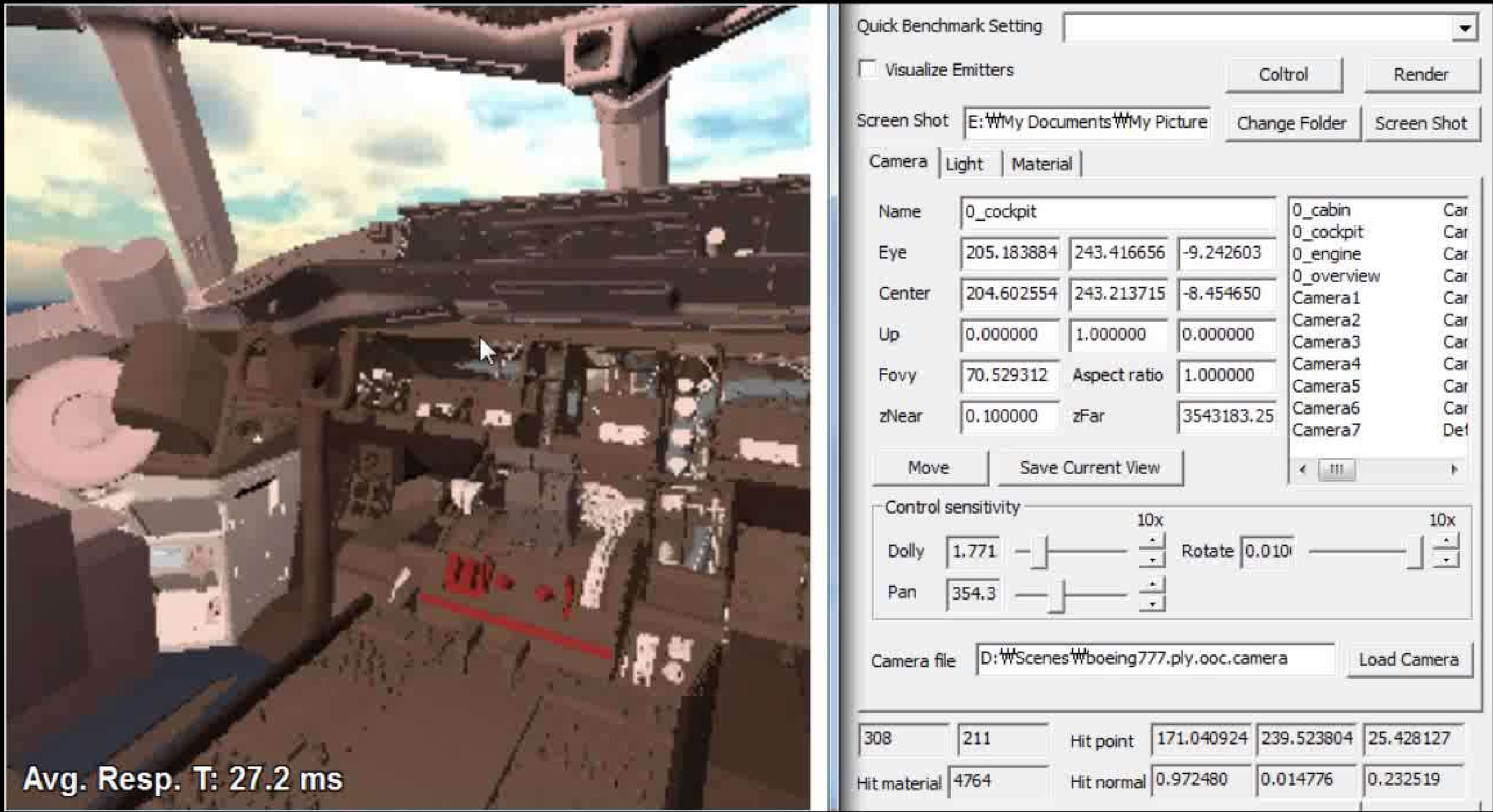


Progressive Rendering

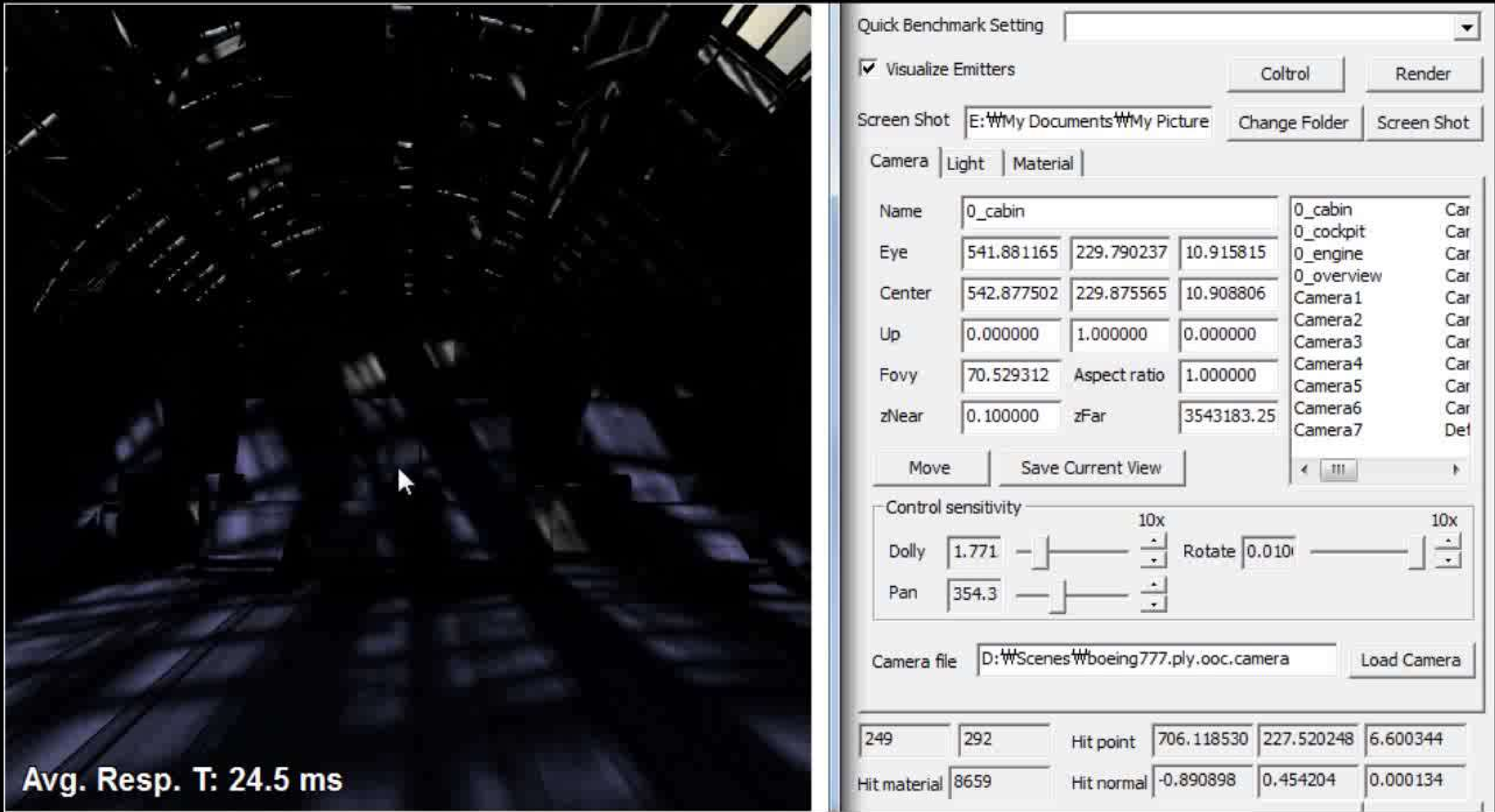
- Progressively refine the frame



Materials Changes



Lights Changes

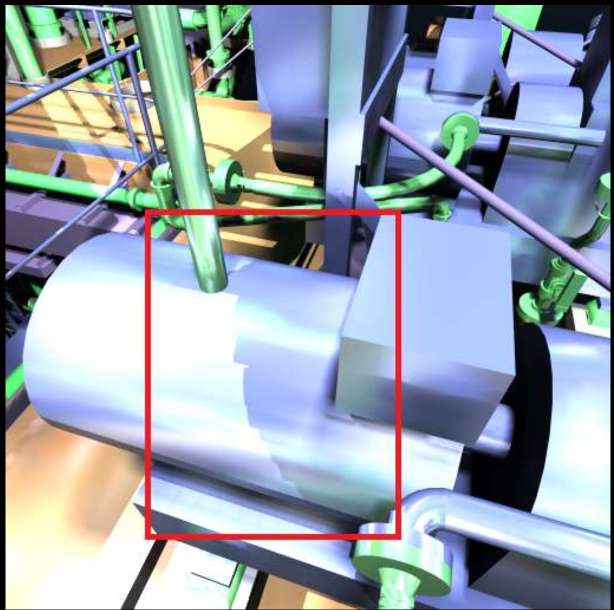


Conclusion

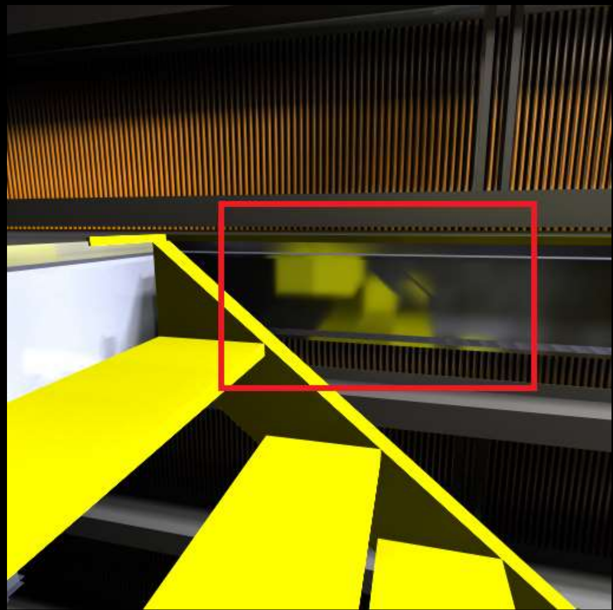
- Present an integrated progressive rendering framework for global illumination of massive models
  - Use a decoupled representation: HCCMeshes in CPU and ASVOs in GPU for handling large-scale models
- Reduce expensive transmission costs and achieve high utilizations for CPU and GPU

Limitations

- Volumetric representation
  - Biased and inconsistent
  - Spans more space than its geometric model



Point light sources



Highly glossy materials

High-Performance Graphics 2017

Los Angeles | July 28–30, 2017

TIMELINE SCHEDULING FOR OUT-OF-CORE RAY BATCHING

Myungbae Son Sung-Eui Yoon

SGVR Lab
KAIST

KAIST School of Computing

Our Scenario

- Complex scenes
  - Out-of-core model: Too big data!
  - Cannot be stored in main / GPU memory

Boeing 777, 366 M tri. (20 GB)

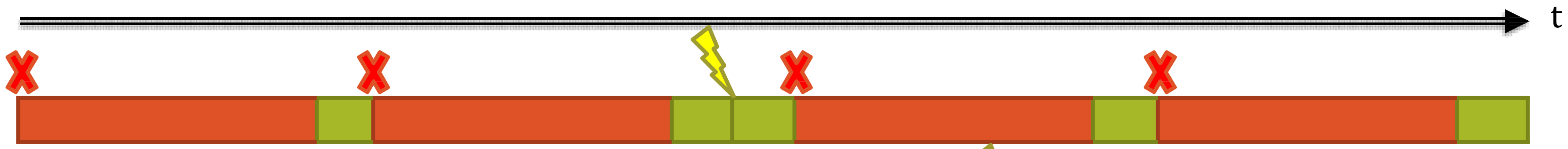
- Complex device configurations
  - Distributed memory cluster system
  - Client-assisted remote rendering
  - Renderfarm of heterogeneous devices

KAIST School of Computing

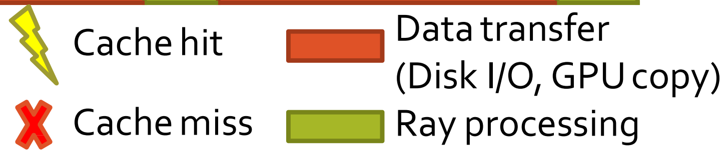
SVGR

Challenges

- Massively complex scene
  - Over **96%** of runtime is spent on I/O in naïve BDPT (Boeing777)



- Excessive page swap required
- I/O cost dominates the rendering time

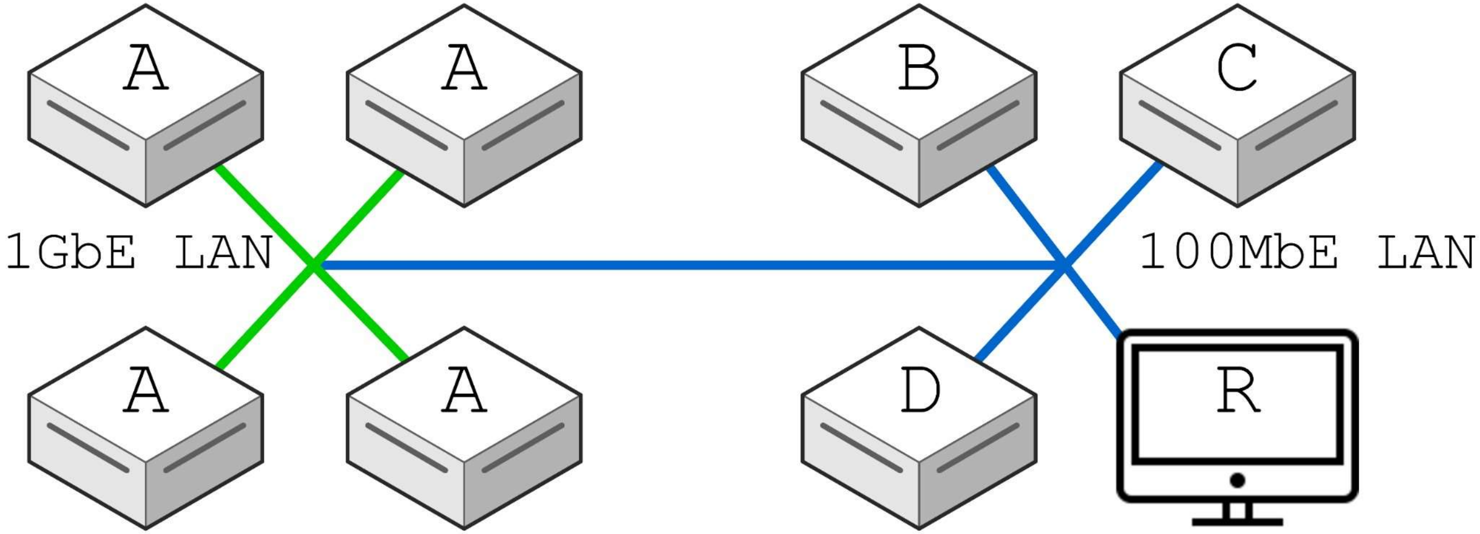


- Global Illumination with incoherent rays
  - Efficient ray scheduling is required

KAIST School of Computing

Challenges

Complex and heterogenenous device configurations…



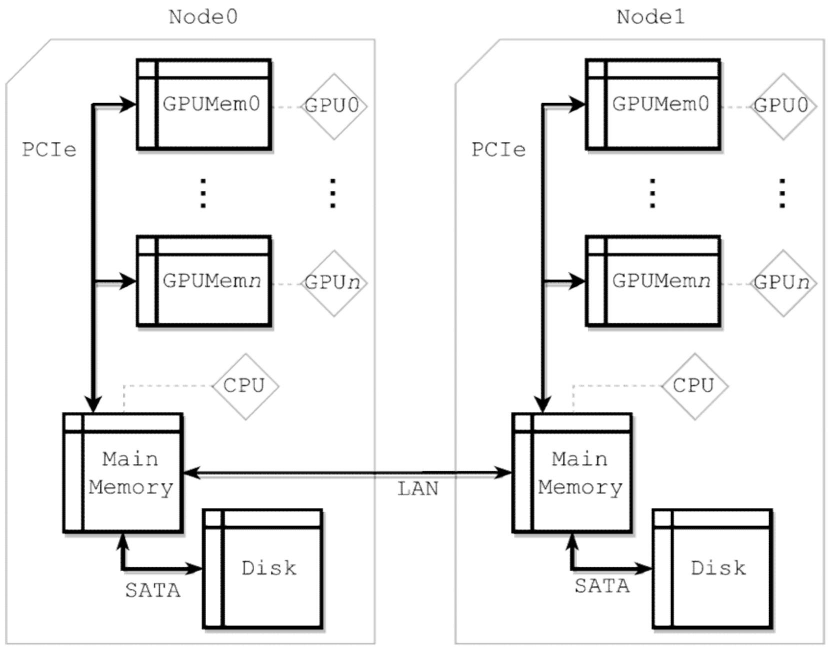
KAIST School of Computing

SVRG

Challenges

Further down to the processor and memory hierarchy level…

- Different **processors**
- Different **memory channels**
- Different **nodes** and **network**



Goal & Contributions

Design a scheduler for global illumination

- Processes massive models
- Supports variety of computing environments
  - Complex and heterogeneous device configurations

Our contributions

- A modeling technique: device configurations and jobs
- A scheduling algorithm: Greedy Makespan Balancing (GMB)
- An adaptation to path tracer

KAIST School of Computing

SVRG

OUR APPROACH

Our Approach

- **Formulation technique for MC ray tracing jobs**
  Device Connectivity Graph (DCG) and Timing Model

- Timeline scheduling and Greedy Makespan Balancing algorithm
  Simple, iterative algorithm that considers utilization and latency hiding

- Adaptation to actual renderer framework
  Out-of-core path tracer

Formulation: Device Connectivity Graph

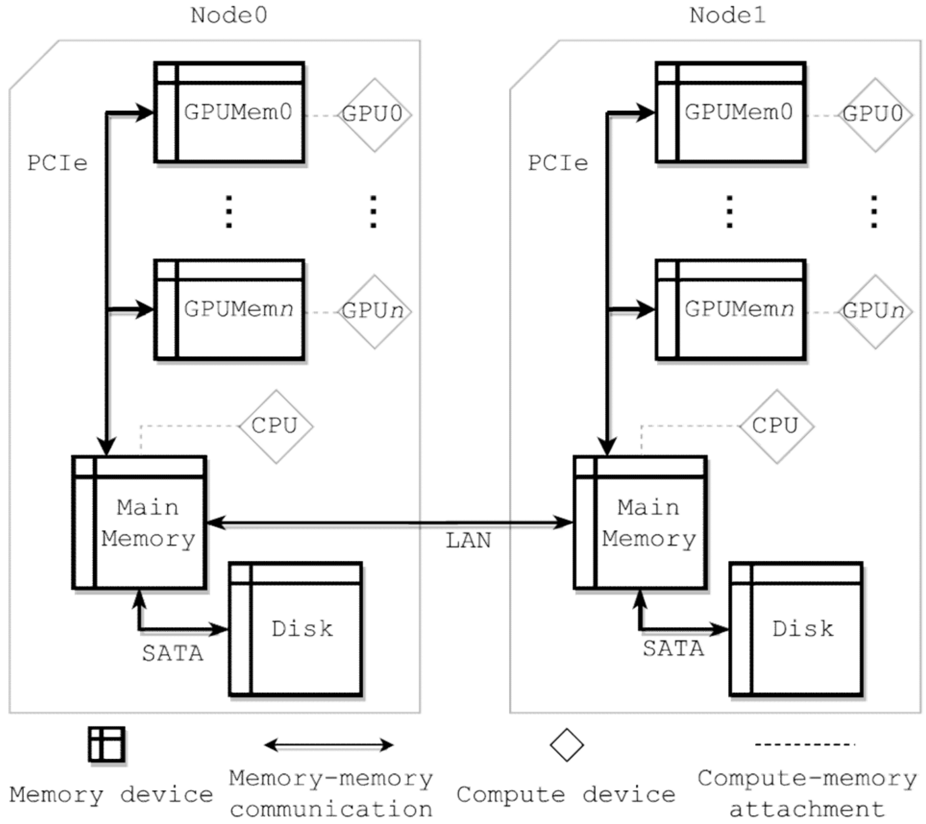
- Graph of memory devices
  - Memory

    Disk storage, RAM, GMEM

  - Connections (Channels)

    PCIe (RAM ↔ GMEM)
    SATA (Disk ↔ RAM)
    LAN (RAM ↔ RAM)
    …

- Stores bandwidth information



Formulation: Timing Model

- Assume simple yet efficient linear model on time
  - Job execution

$$T_{EXEC}(d, j, W) = \begin{cases} 0, & if\,W = \emptyset \\ T_{SETUP}(d, j) \\ \quad + T_{RATE}(d, j) \cdot (|w_1|, |w_2|, \ldots)', & otherwise \end{cases}$$

  - Data transfer

$$T_{TRANS}(d_i \rightarrow d_j, w) = T_{LAT}(d_i \rightarrow d_j) + \frac{|w|}{T_{BW}(d_i \rightarrow d_j)}$$

- Fitting each parameter ($T_{SETUP}, T_{RATE}, T_{LAT}, T_{BW}$)
  - Use least squares method on test run

Our Approach

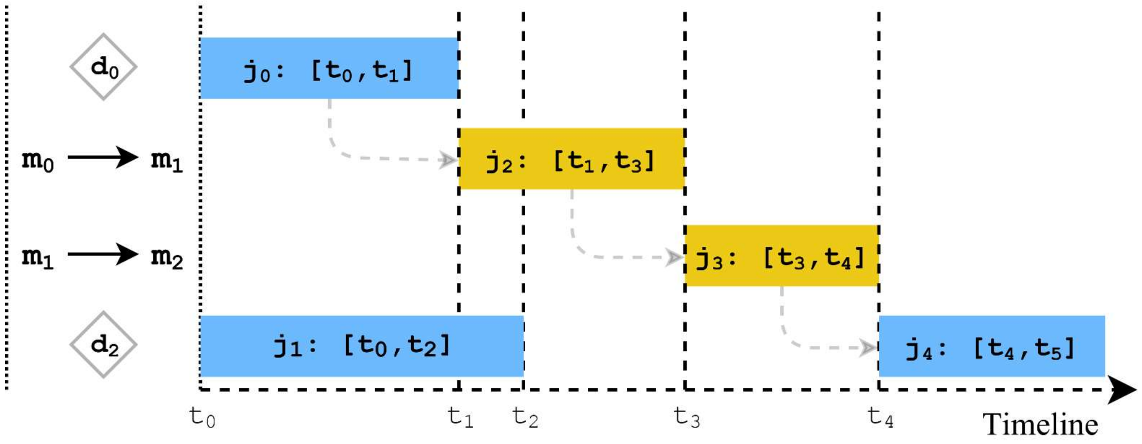
- Formulation technique for MC ray tracing jobs
  Device Connectivity Graph (DCG) and Timing Model

- **Timeline scheduling and Greedy Makespan Balancing algorithm**
  Simple, iterative algorithm that considers utilization and latency hiding

- Adaptation to actual renderer framework
  Out-of-core path tracer

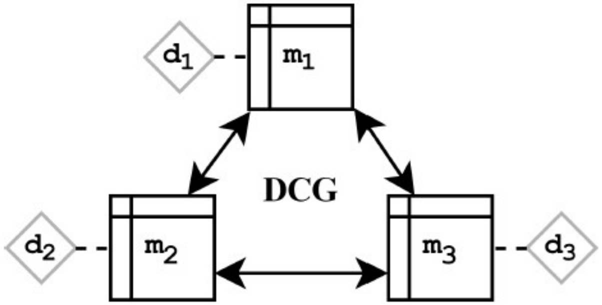
Timeline Scheduling

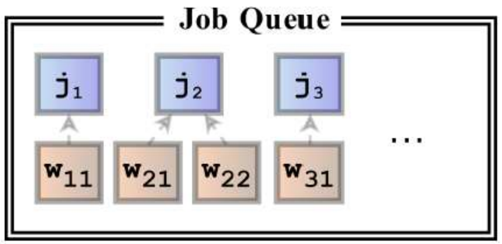
- A representation of schedule with timing constraints
  - For ◇ processors
    Executable jobs are allocated
  - For ↔ memory channels
    Data transfers are allocated
  - Dependencies
    between jobs and fetches

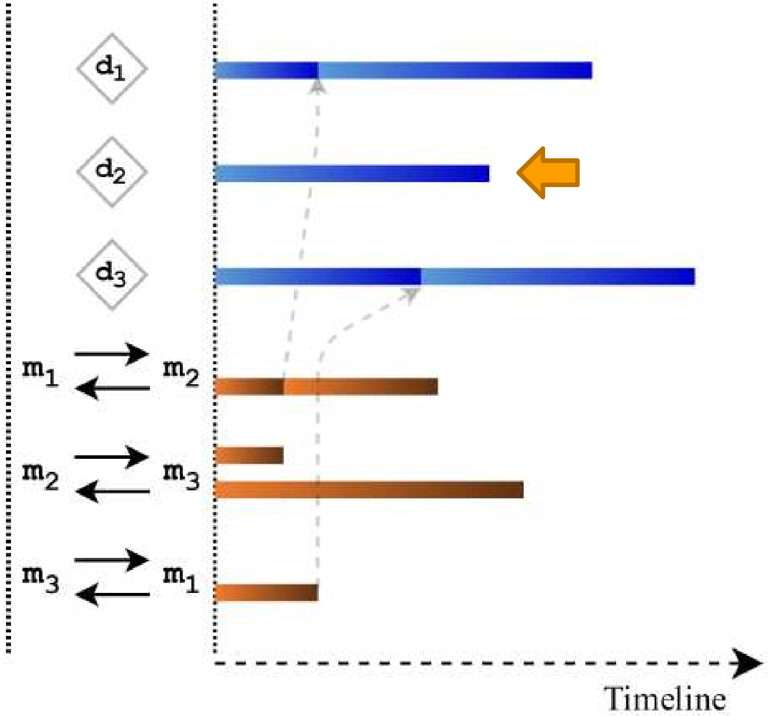


Def. schedule: a set of timelines that jobs and fetches are allocated

Greedy Makespan Balancing Algorithm



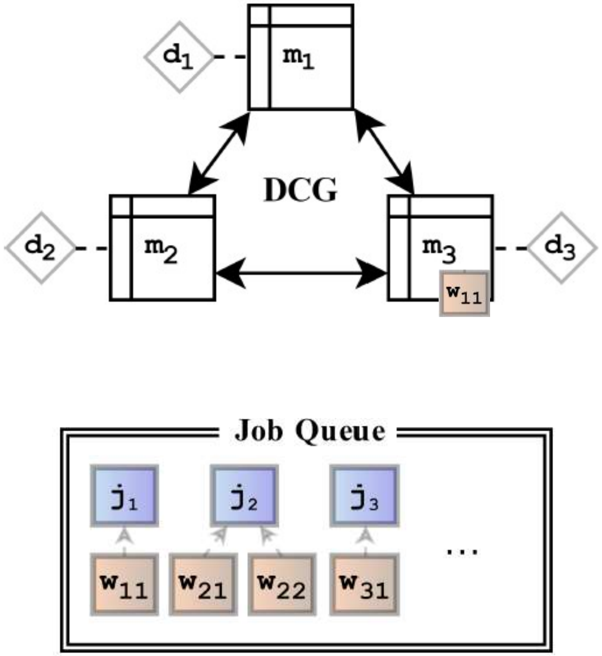


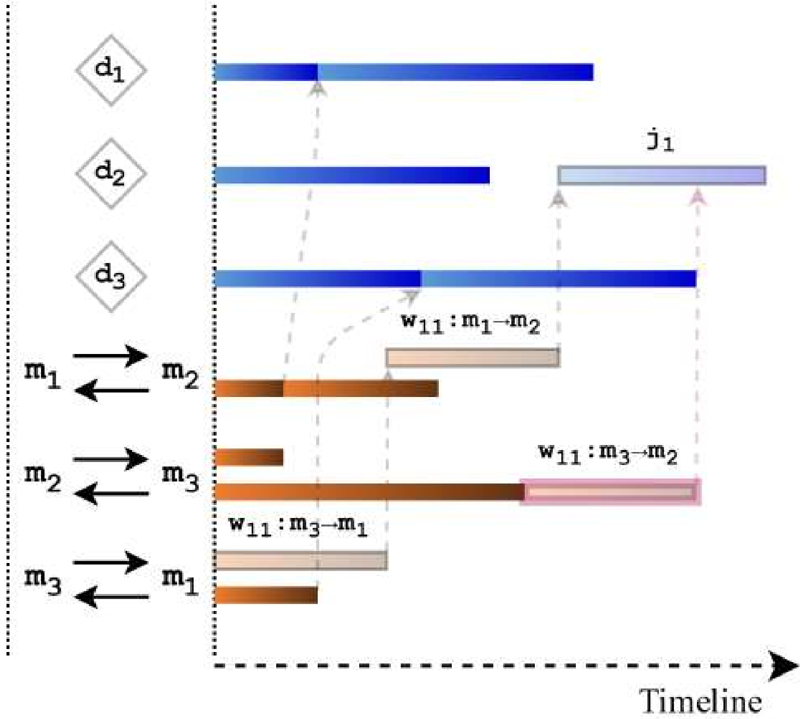


KAIST School of Computing

1. Choose least occupied compute device d

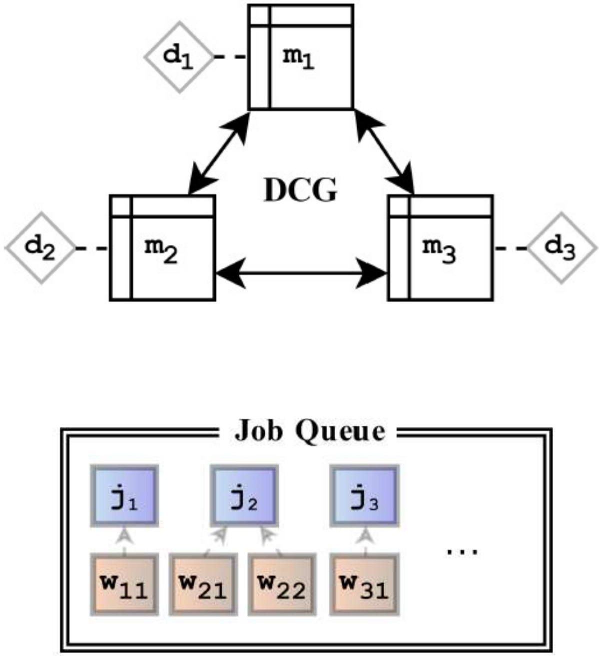
Greedy Makespan Balancing Algorithm

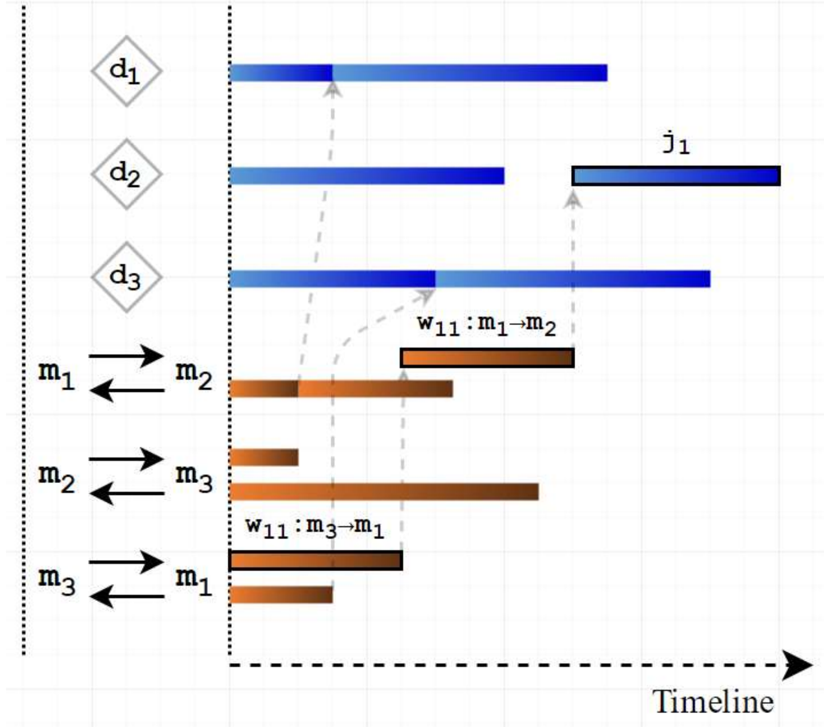




2. Find job j_i that can be run at d as soon as possible

Greedy Makespan Balancing Algorithm





4. Repeat until devices are occupied enough

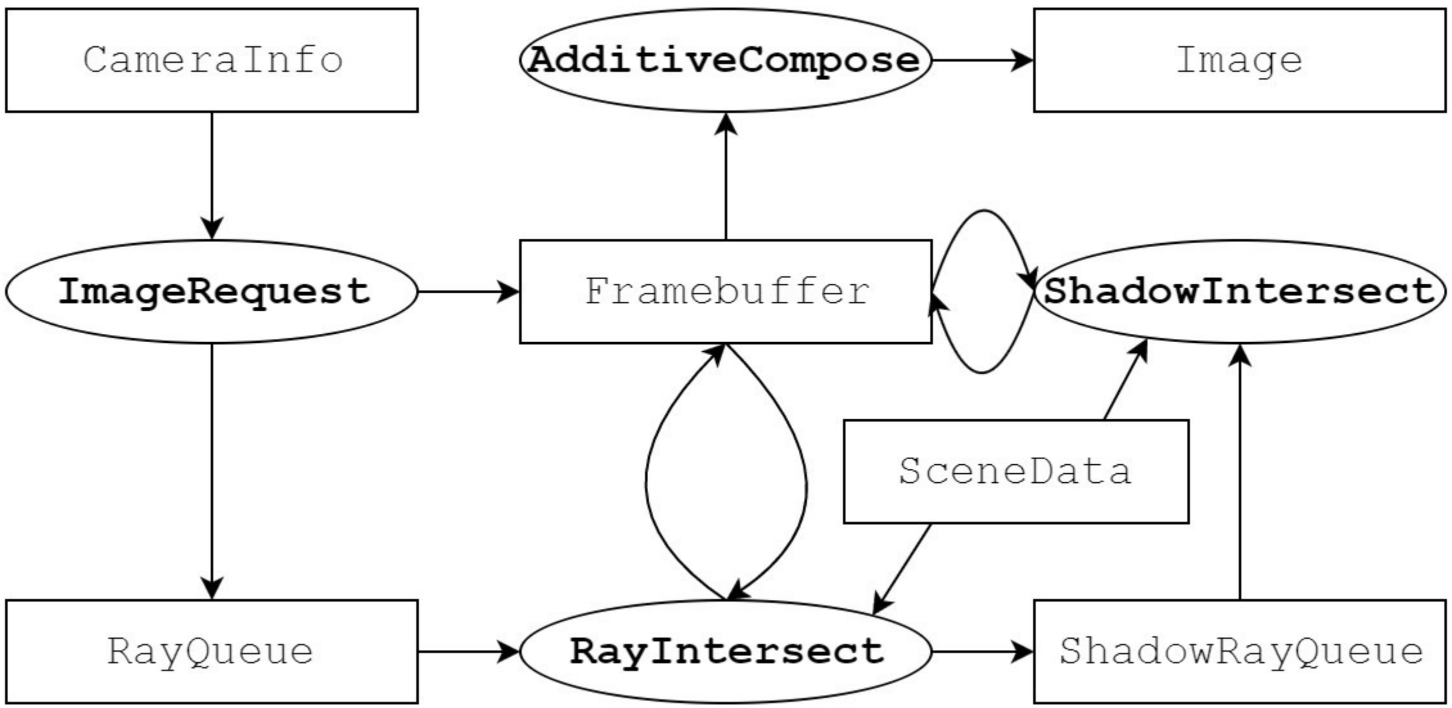
Our Approach

- Formulation technique for MC ray tracing jobs
  Device Connectivity Graph (DCG) andTiming Model

- Timeline scheduling and Greedy Makespan Balancing algorithm
  Simple, iterative algorithm that considers utilization and latency hiding

- **Adaptation to actual renderer framework**
  Out-of-core path tracer

Out-of-core Path Tracer Jobs



KAIST School of Computing

RESULTS

Benchmark scene

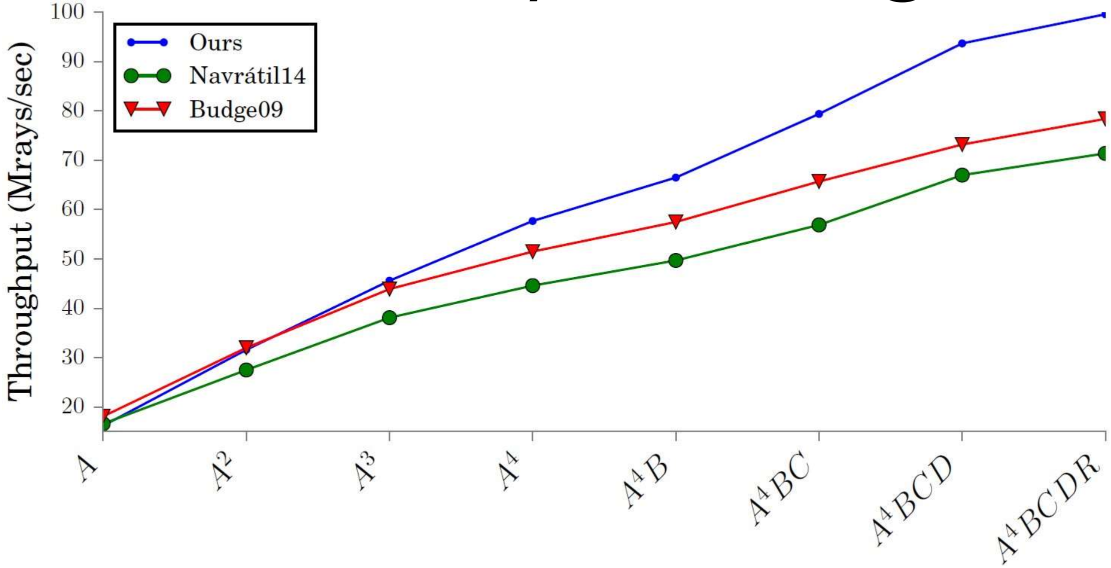
Boeing777 (26.5GB, 496M tri, 5.2sec/img)

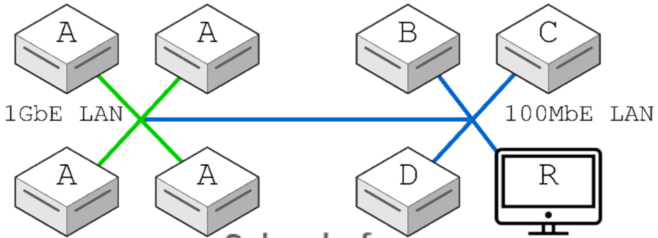
SponzaMuseum (12.3GB, 245M tri, 34.8 sec/img)

$(800 \times 800 \times 32spp \times 60frames)$

- Model preparation
  - Even-sized median-split kdtree, 2^7 / 2^6 subdivision, respectively

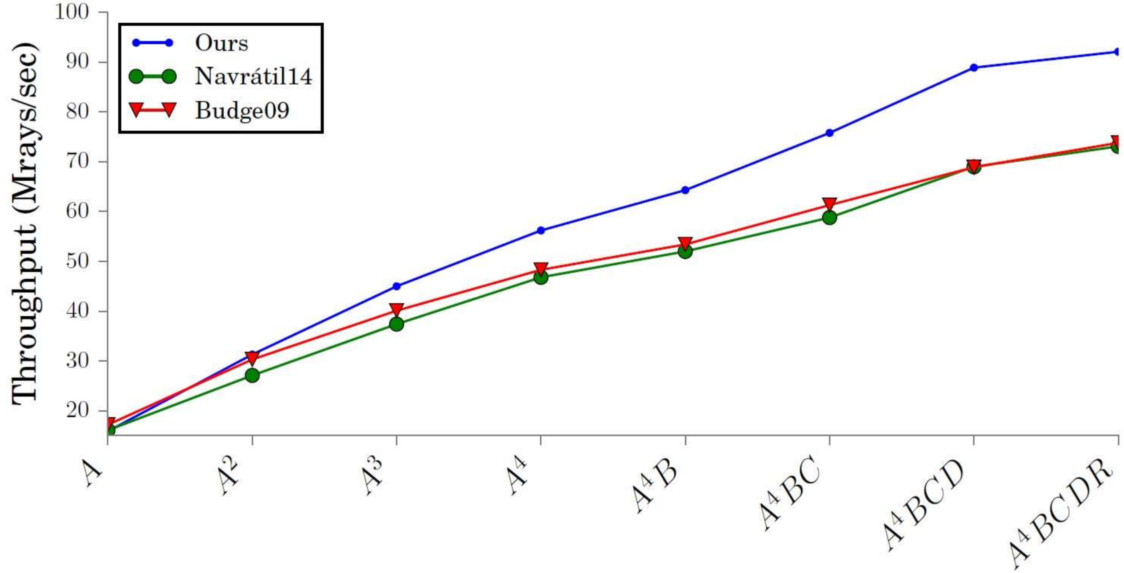
Horizontal Scalability – Boeing777

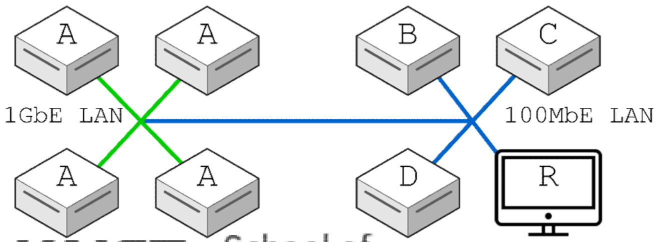




Type	CPU	Main memory	GPU Memory	GPU	Note
A	i7-4770K 3.5GHz octa-core	DDR3 8GB	6GB	GTX Titan	1GbE LAN, 4 nodes
B	i7-4790K 4GHz octa-core	DDR3 8GB	6GB	GTX Titan	
C	Xeon E5-2690 2.9GHz 16-core	DDR3 8GB	6GB	GTX Titan	
D	Xeon E5-2690 2.6GHz 16-core	DDR3 8GB	6GB	GTX Titan X	
R	i7-3770k 3.5GHz quad-core	DDR3 8GB	4GB	GTX980	

Horizontal Scalability – SponzaMuseum



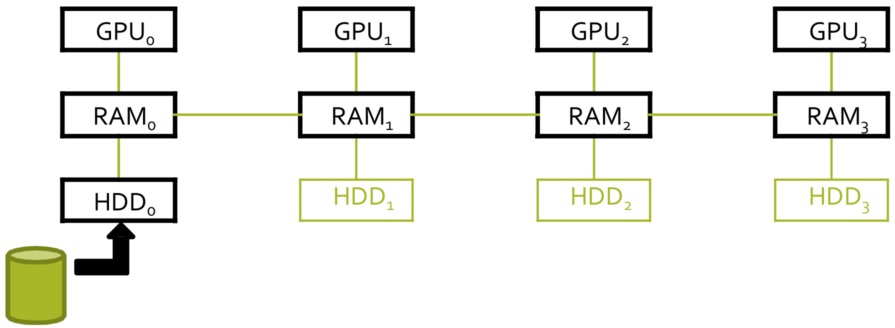


Type	CPU	Main memory	GPU Memory	GPU	Note
A	i7-4770K 3.5GHz octa-core	DDR3 8GB	6GB	GTX Titan	1GbE LAN, 4 nodes
B	i7-4790K 4GHz octa-core	DDR3 8GB	6GB	GTX Titan	
C	Xeon E5-2690 2.9GHz 16-core	DDR3 8GB	6GB	GTX Titan	
D	Xeon E5-2690 2.6GHz 16-core	DDR3 8GB	6GB	GTX Titan X	
R	i7-3770k 3.5GHz quad-core	DDR3 8GB	4GB	GTX980	

KAIST School of Computing

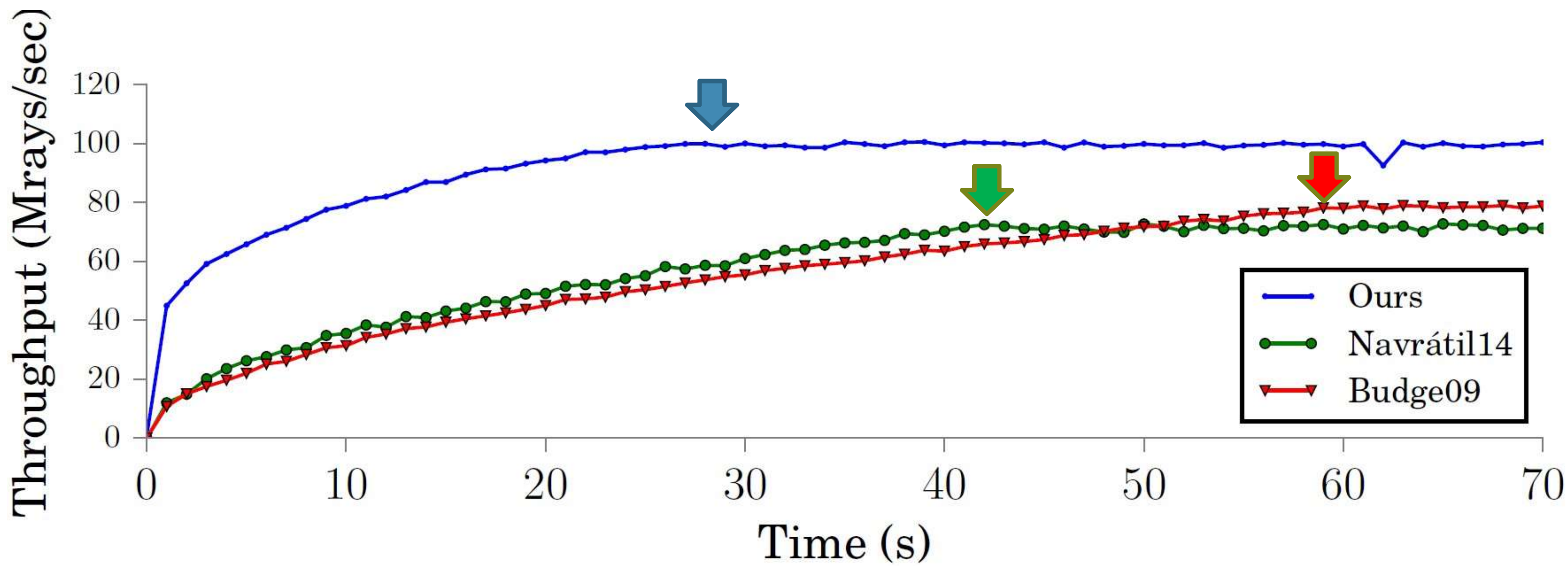
Efficiency on Data Fetching

- Central scene DB scenario



- Initially no data at slave nodes at all
- The master node gives scene data blocks on-demand

Efficiency on Data Fetching



(Boeing777)

Our method converges to peak performance much faster than previous methods

Conclusion

- Presented specification techniques for out-of-core MC ray tracing on arbitrary hardware setup
  - DCG and timing model
- Presented a timeline based scheduling algorithm
  - GMB algorithm
- Applied to the out-of-core path tracer
  - Prediction technique for future rays

Conclusions

- Two different techniques, manual assignment and automatic approaches, for large-scale rendering
- Released a free book on rendering

- Working on a journal version of our tutorial