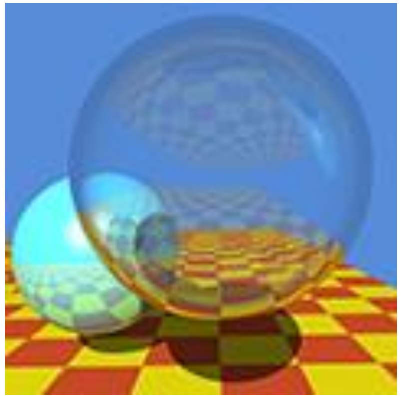
SATO: Surface-Area Traversal Order for Ray Tracing

SEP 13, 2013

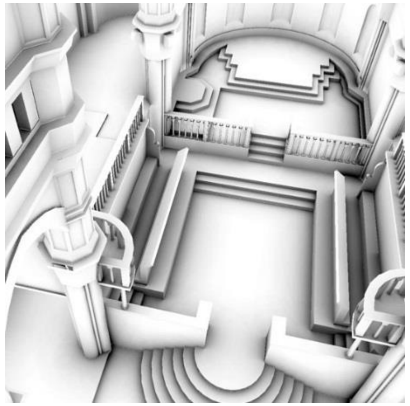
JAE-HO NAH

Two Ray Types in Ray Tracing

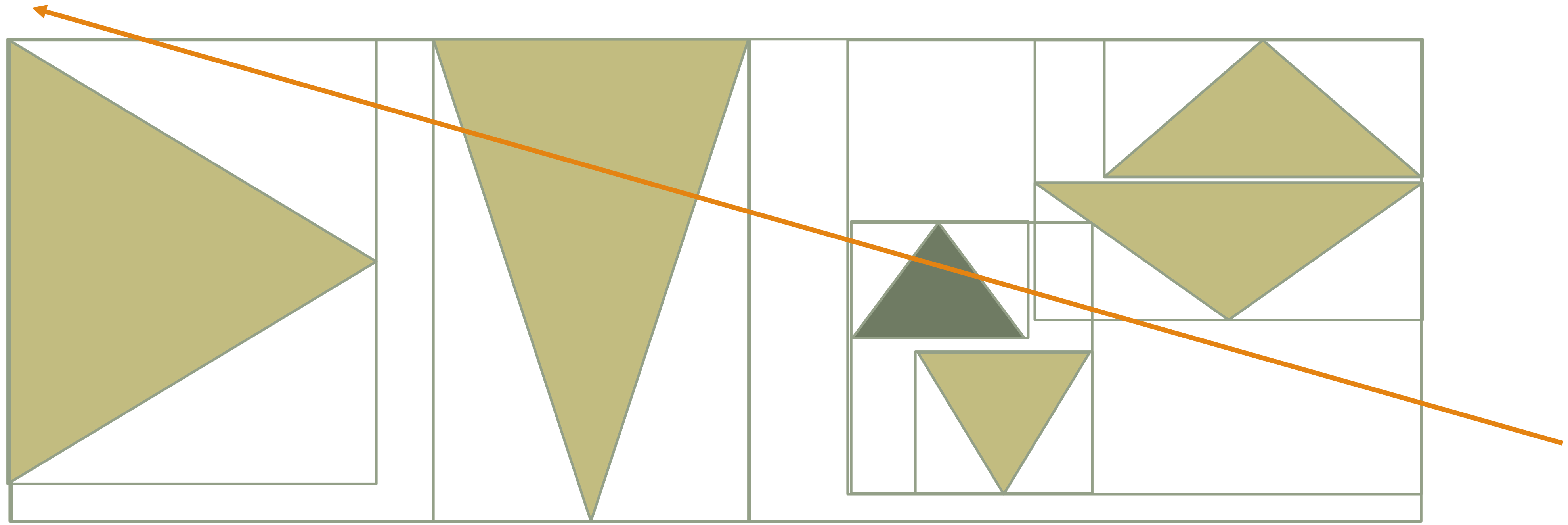
- Radiance ray
  - For primary visibility, reflection, refraction, etc.



- Shadow ray (occlusion ray)
  - For hard and soft shadows, ambient occlusion, etc.

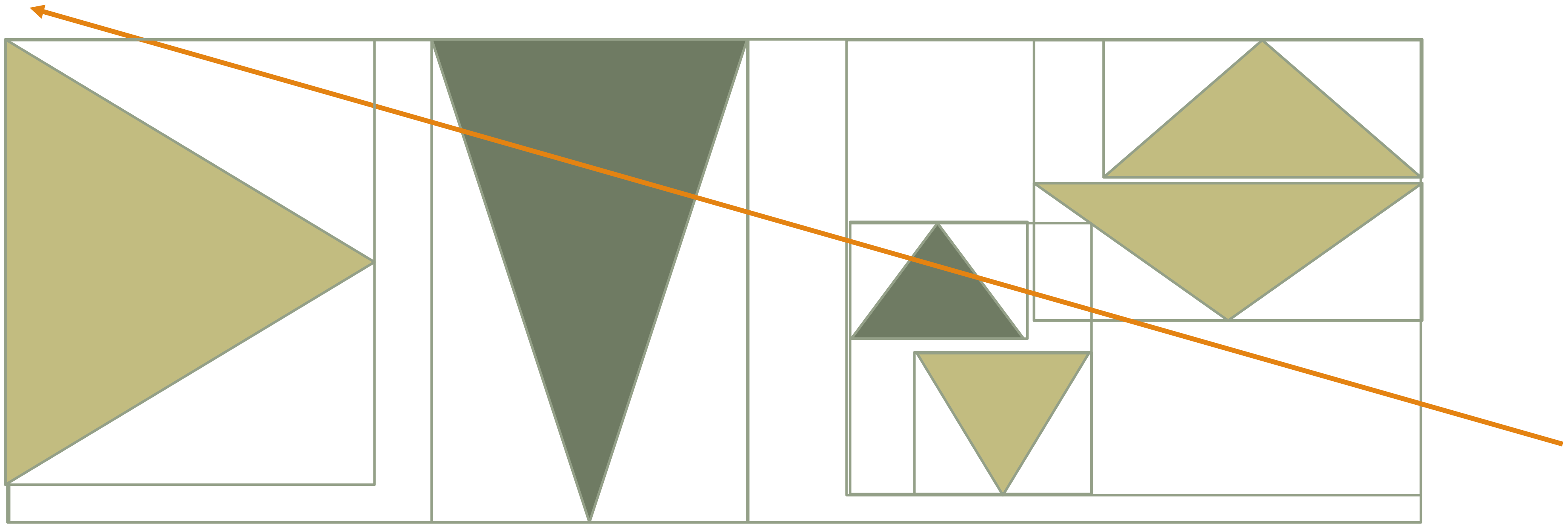


Traversal of a Radiance Ray



- Need for the closest hit point
- Front-to-back traversal order is ideal

Traversal of a Shadow Ray



- Any hit point is possible
- Front-to-back traversal order may not be ideal

Surface-Area Heuristic
[Goldsmith and Salmon 1987]

- Standard tree construction method
- Greedy SAH construction determines the optimal split position by using the following equation:

$$C_V(p) \approx K_T + K_I \left(\frac{SA(V_L)}{SA(V)} T_L + \frac{SA(V_R)}{SA(V)} T_R \right)$$

SA (v) = surface area of V

T_V = # of tris in V

Expected cost of the voxel V

Traversal cost

Inter-section cost

Left node's cost

Right node's cost

Surface-Area Heuristic
[Goldsmith and Salmon 1987]

- Standard tree construction method
- Greedy SAH construction determines the optimal split position by using the following equation:

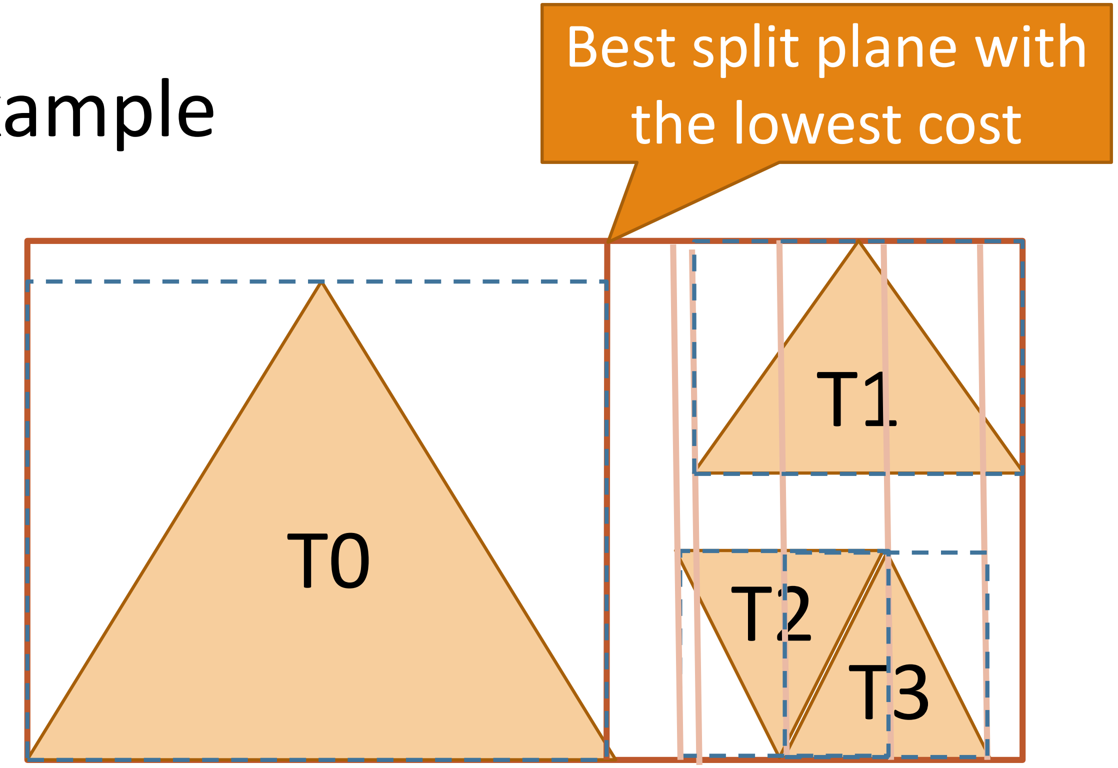
$$C_V(p) \approx K_T + K_I \left(\frac{SA(V_L)}{SA(V)} T_L + \frac{SA(V_R)}{SA(V)} T_R \right)$$

SA (v) = surface area of V
T_V = # of tris in V

- Assumption: the probability of a node being pierced by rays is proportional to the node’s surface area with finite random rays`

Surface-Area Heuristic
[Goldsmith and Salmon 1987]

- Example



State-of-the-Art Shadow-Ray Traversal Order

- RTSAH (Ray Termination Surface-Area Heuristic) [Ize and Hansen 2011]
  - Calculates the expected traversal cost of each child node (preprocessing)

$$C_{\text{node}} = Min(leftFirst, rightFirst) \quad (6)$$

$$leftFirst = T_{\text{step}} + P_l C_l + (P_{jr} + P_{lr} V_l)(T_{\text{step}} + C_r) + P_e T_{\text{step}} \quad (3)$$

$$rightFirst = T_{\text{step}} + P_r C_r + (P_{jl} + P_{lr} V_r)(T_{\text{step}} + C_l) + P_e T_{\text{step}} \quad (4)$$

$$V_{\text{node}} = \begin{cases} P_{jl} V_l + P_{jr} V_r + P_{lr} V_l V_r + P_e & \text{if an inner node,} \\ 0 & \text{if nonempty leaf,} \\ 1 & \text{if empty leaf.} \end{cases} \quad (7)$$

  - The cheaper child node is first visited in ray traversal
  - Up to 2X speedup

State-of-the-Art Shadow-Ray Traversal Order

- SRDH (Shadow Ray Distribution Heuristic) [Feltman et al. 2012]
  - Shadow-ray-specialized BVH construction method
  - Traversal results of a small representative set of rays are used for both BVH construction and traversal-order determination
  - Frequent occluders are located in upper-level nodes and are first visited
  - 22~56% less traversal steps than SAH-constructed trees

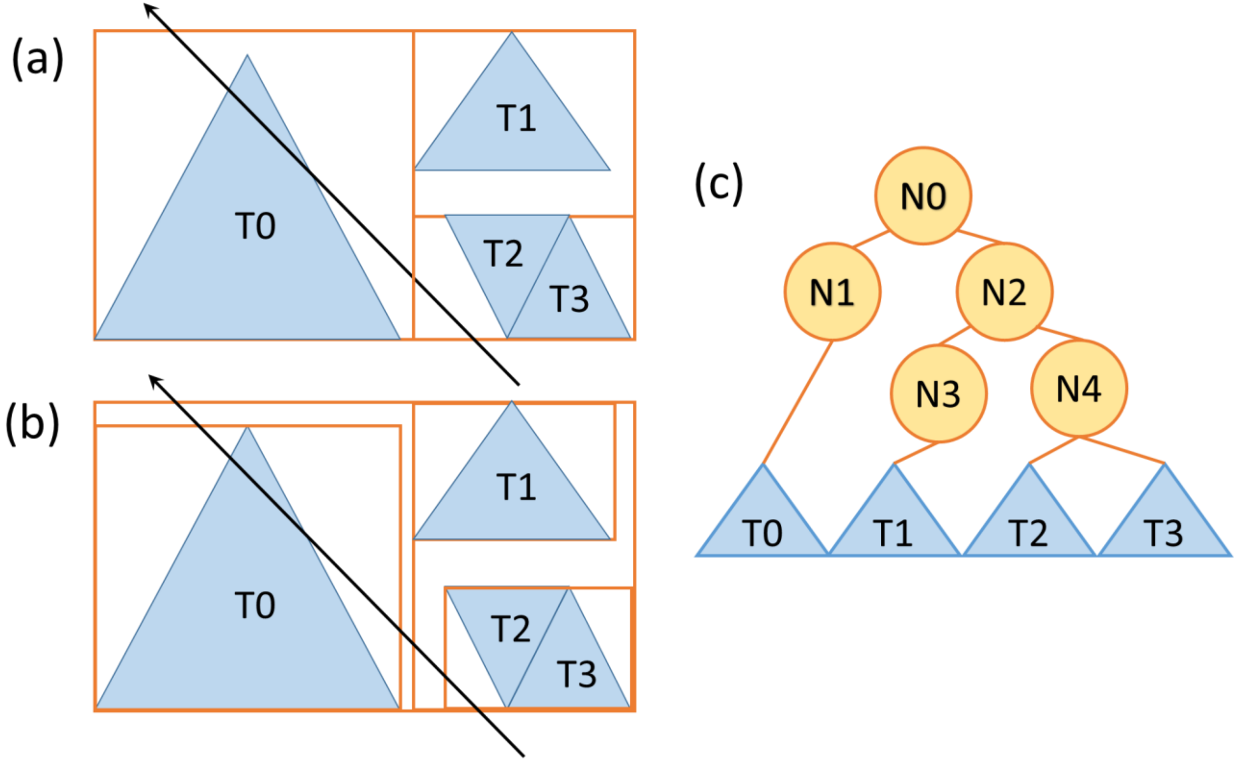
State-of-the-Art Shadow-Ray Traversal Order

- Limitation of RTSAH and SRDH: preprocessing cost
  - RTSAH: ~51ms (fast approximate RTSAH), ~5.8s (RTSAH)
  - SRDH: ~20s
  - May not be suitable for dynamic scenes
- How can we quickly determine efficient traversal order?

Surface-Area Traversal Order

- Surface-area traversal order (SATO)
  - To determine TO, simply use the surface area (SA)
- Two major goals
  - To minimize the traversal order (TO) calculation time
  - To quickly find a large occluder for shadow ray tracing
- Three sub-metrics of SATO
  - NodeSATO: uses each node's SA
  - PrimSATO: uses average or maximum SA of each primitive in a node
  - PrimNumTO: uses primitive numbers

Example of Traversal Order



(a) kd-tree, (b) BVH, (c) tree representation

- Front-to-back
  - N0→ N2→N4→T2→T3→N1→T0
- NodeSATO and PrimSATO
  - N0→ N1→T0
  - N1 is first visited instead of N2

Why does SATO work?

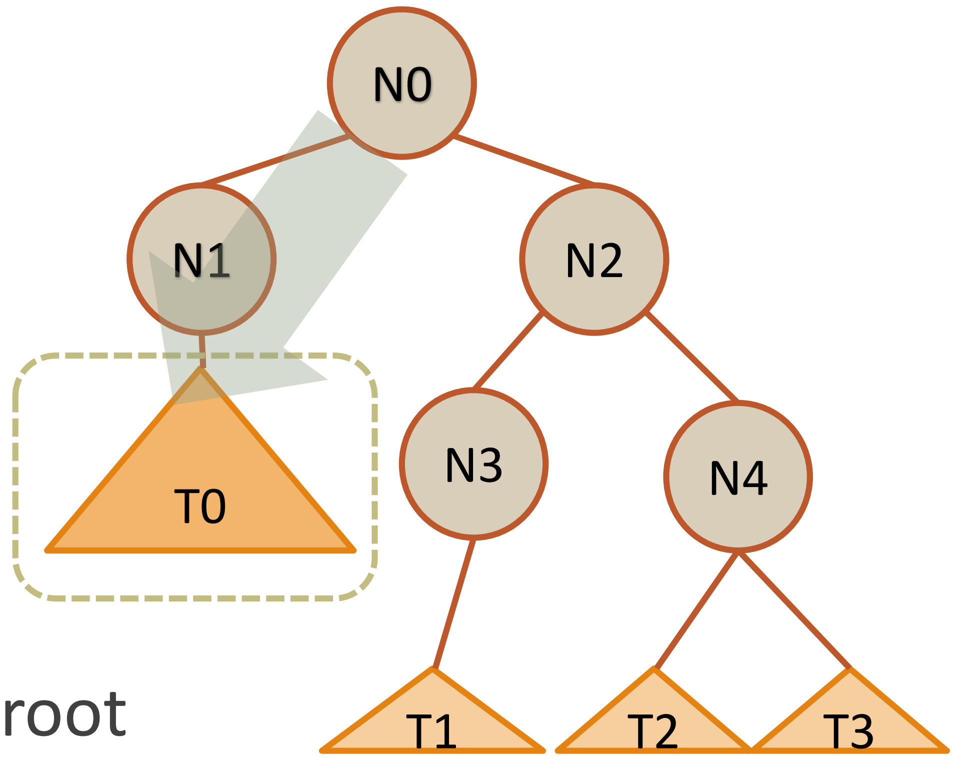
- Three assumptions will be introduced for

: PrimSATO, NodeSATO in kd-trees, and NodeSATO in BVHs

- We will focus on NodeSATO, but the relationship between PrimSATO and NodeSATO is important to verify NodeSATO

First Assumption for PrimSATO

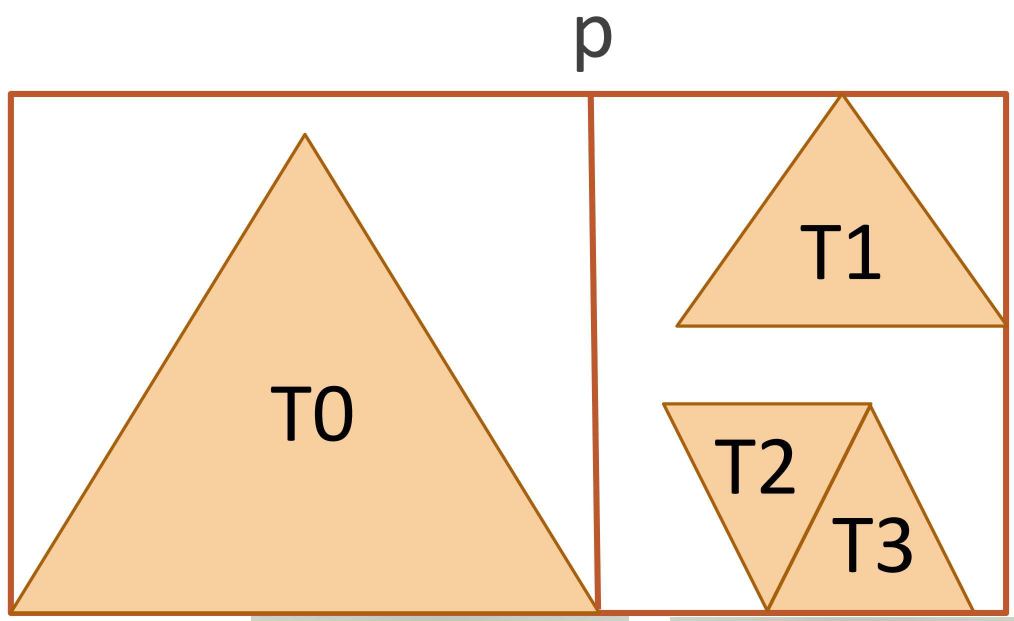
- Large primitives are usually located in the upper-level nodes in a tree, so intersecting with a large primitive first can result in early termination of a shadow ray.
- This assumption is based on the character of SAH-constructed trees
  - The SAH keep larger primitives near the root



Second Assumption for NodeSATO/PrimNumTO in Kd-trees

- In SAH kd-trees, the child node with the larger SA has a higher probability of enclosing larger primitives
- In many cases, the SAH costs of each node (a) (b) are comparable
  - Larger SA ≈ less # of prims

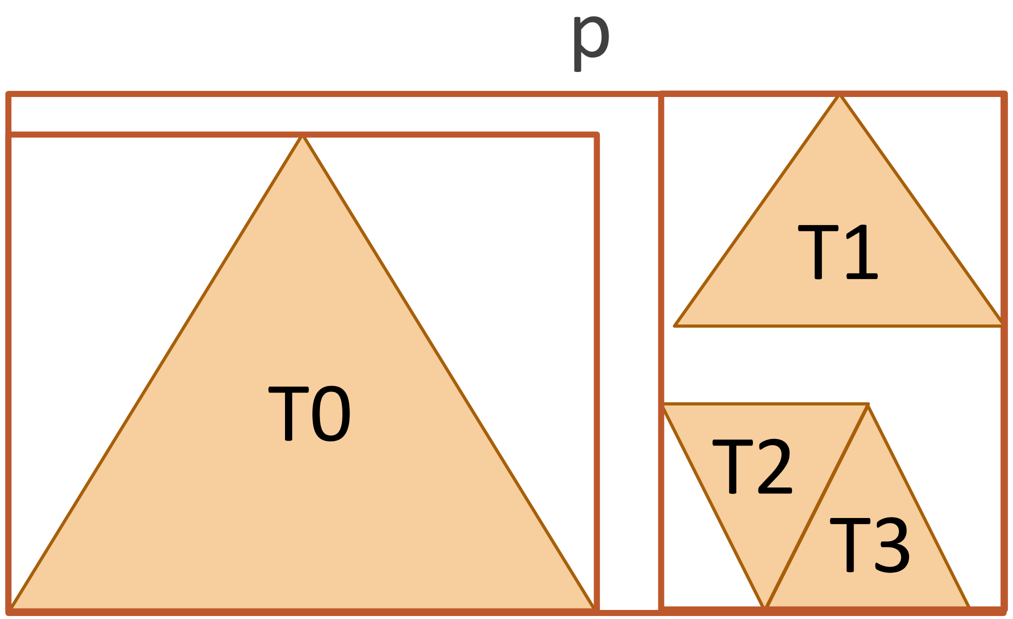
→ Possibility of large empty spaces or large prims



$$C_V(p) \approx \underbrace{SA\,(V_L)\cdot T_L}_{(a)} + \underbrace{SA(V_R)\cdot T_R}_{(b)}$$

Third Assumption for NodeSATO in BVHs

- In BVHs, the child node with the larger SA has a higher probability of enclosing larger primitives
- The BV of a parent BVH node includes the BVs of its primitives
  - Larger SA ≈ larger prims



PrimSATO

- Psuedo code in tree construction

```
find the optimal split plane
if (optimal split plane exists)
  make an inner node
  calculate the avg or max  SA value of prims in each child node
  compare each child's avg or max SA value
  set isLeftCheaper of the node
else
  make a leaf node
```

NodeSATO

- Psuedo code in tree construction

```
find the optimal split plane
if (optimal split plane exists)
  make an inner node
  compare each child node's SA
  set isLeftCheaper of the node
else
  make a leaf node
```

NodeSATO

- NodeSATO can be used with various BVH update methods
- Selective SATO update is possible with the tree-rotation algorithm [Kopta et al. 2012]
- Pseudo code in tree rotation

```
if (a rotation can decrease the SAH cost)
  do a rotation
  compare the SAs of the rotated inner nodes
  set isLeftCheaper of the nodes
```

PrimNumTO

- Psuedo code in tree construction

```
find the optimal split plane
if (optimal split plane exists)
  make an inner node
  compare primitive numbers of each child node
  set isLeftCheaper of the node
else
  make a leaf node
```

SATO Traversal

- Same as the RTSAH traversal
  - refer to the predefined isLeftCheaper flag
- Psuedo code

if (a shadow ray intersects with an inner node)

  front_son = node.isLeftCheaper ? 0 : 1;

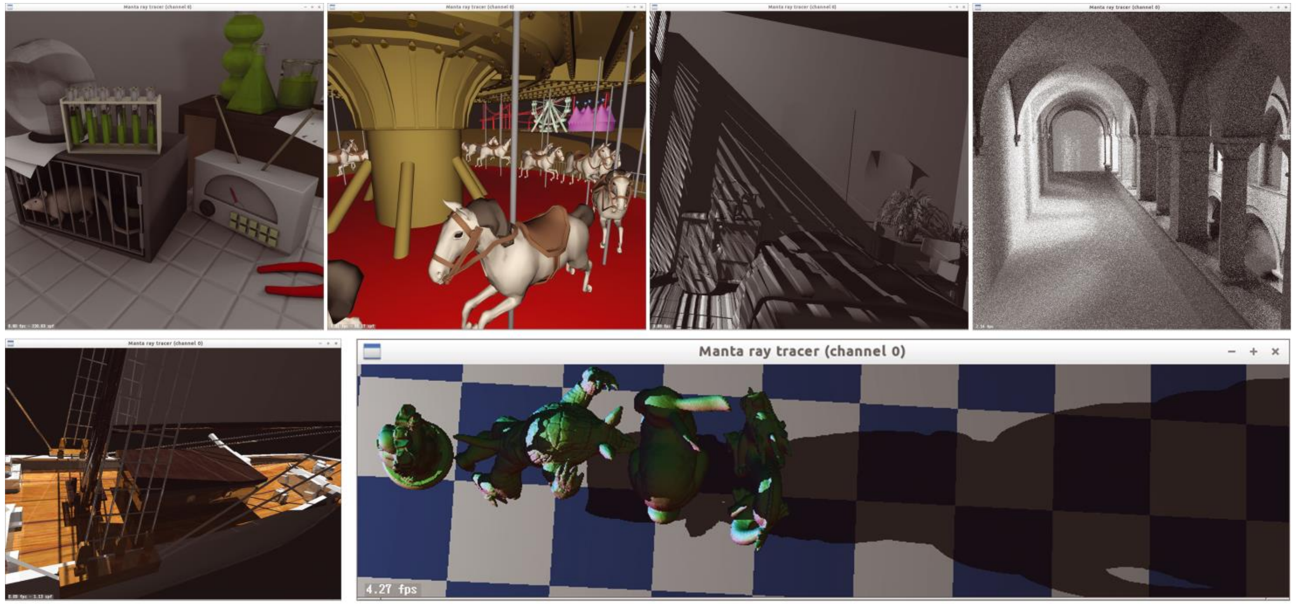
  do an intersection test with node.child+front_son

  do an intersection test with node.child+1-front_son

Experimental Setup

- Intel 3.4GHz Core i7 with 8GB RAM (w/ hyperthreading)
- Manta interactive ray tracer
- Traversal algorithm
  - 8x8 packetized BVH traversal [Wald et al. 2007]
  - Single-ray BVH traversal
  - Single-ray kd-tree traversal
- Traversal order
  - Front-to-back
  - Random
  - RTSAH
  - Approximate RTSAH
  - PrimSATO_{AVG}
  - PrimSATO_{MAX}
  - NodeSATO
  - PrimNumTO

Static Benchmark Scenes



- Mad science (80k tris)
- Carnival (446k tris)
- Bedroom (361k tris)
- Sponza (66k tris)
- Ship (4k tris)
- Shadow overlap (2M tris)

Results in Static Scenes

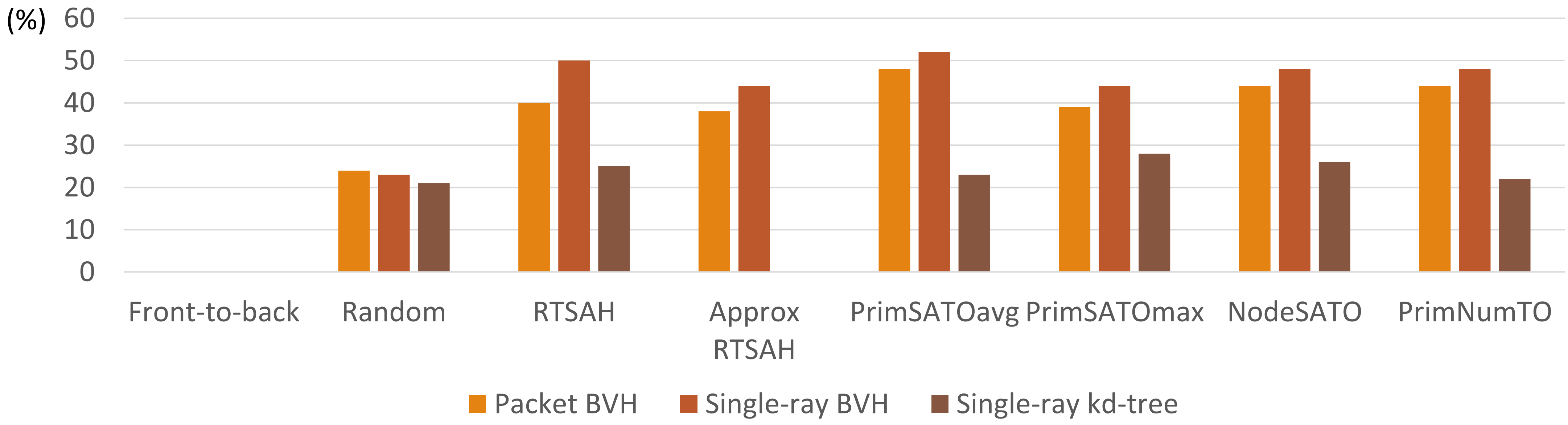
- Traversal order calculation time (w/ a single thread)
  - BVH: NodeSATO/PrimNumTO (~3ms), PrimSATO (~154ms)
  - Kd-tree: NodeSATO/PrimNumTO (~3ms), PrimSATO (~106ms)

Table 1: *TO calculation time for a BVH and a KD-tree (unit: millisecond, lower is better). A single thread was used.*

	BVH						kd-tree				
	RTSAH	Approx RTSAH	PrimSATO AVG	PrimSATO MAX	NodeSATO	PrimNum	RTSAH	PrimSATO AVG	PrimSATO MAX	NodeSATO	PrimNum
Mad Science	28.8	0.9	2.3	2.1	<0.1	<0.1	10.3	5.3	5.1	0.2	0.3
Carnival	185.3	5.4	19.6	18.5	0.6	0.2	39.9	25.8	23.1	1.4	1.6
Bedroom	135.7	3.7	12.0	10.9	0.4	0.1	53.2	33.7	32.1	0.8	1.2
Shadow Overlap	752.7	29.2	154.2	145.0	3.1	1.0	173.8	106.3	103.8	2.8	3.4
Sponza	18.4	0.7	1.8	1.6	<0.1	<0.1	9.0	2.9	2.2	0.3	0.3
Ship	1.3	<0.1	<0.1	<0.1	<0.1	<0.1	0.7	0.2	0.2	<0.1	<0.1

Results in Static Scenes

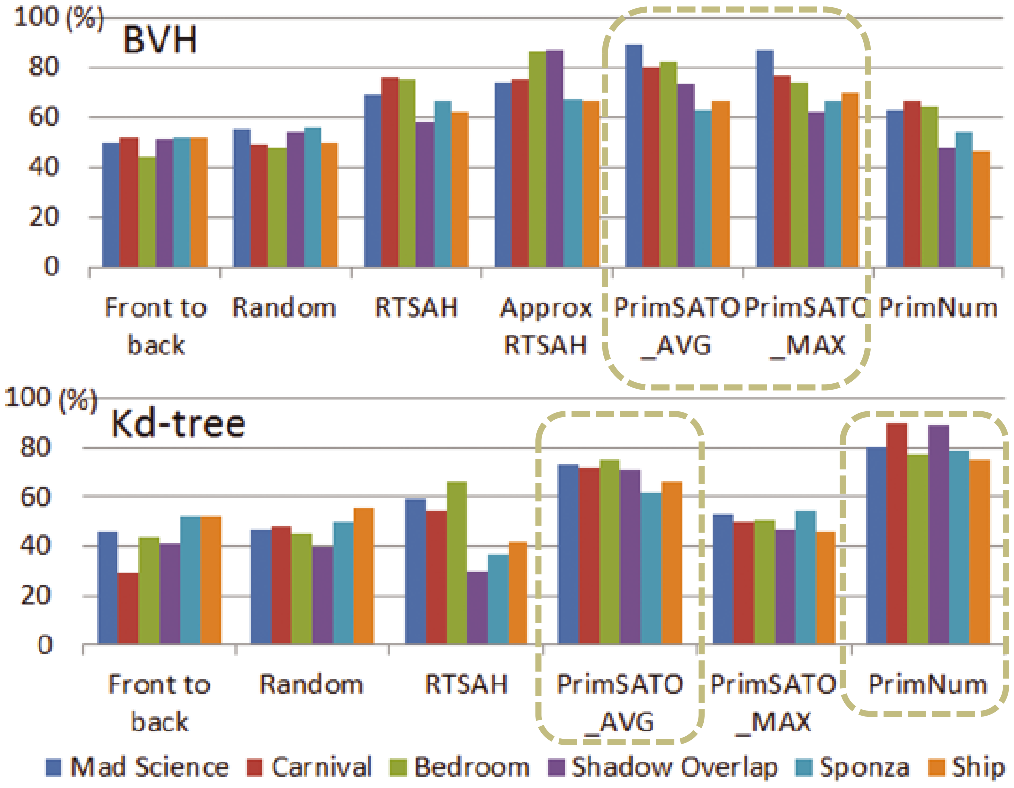
- Rendering performance improvements (in average)



- PrimSATO – up to 8% higher speedup than RTSAH
- NodeSATO/PrimNumTO – similar speedup compared to RTSAH

Results in Static Scenes

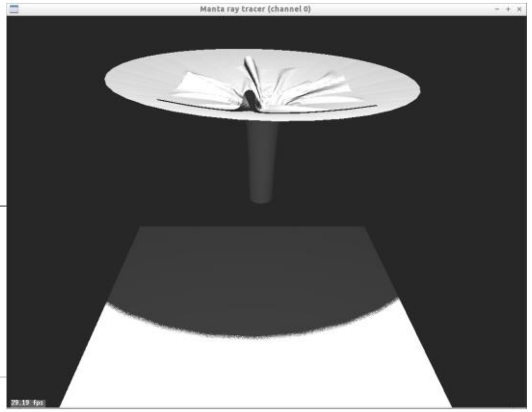
- Traversal-order similarity between NodeSATO and others

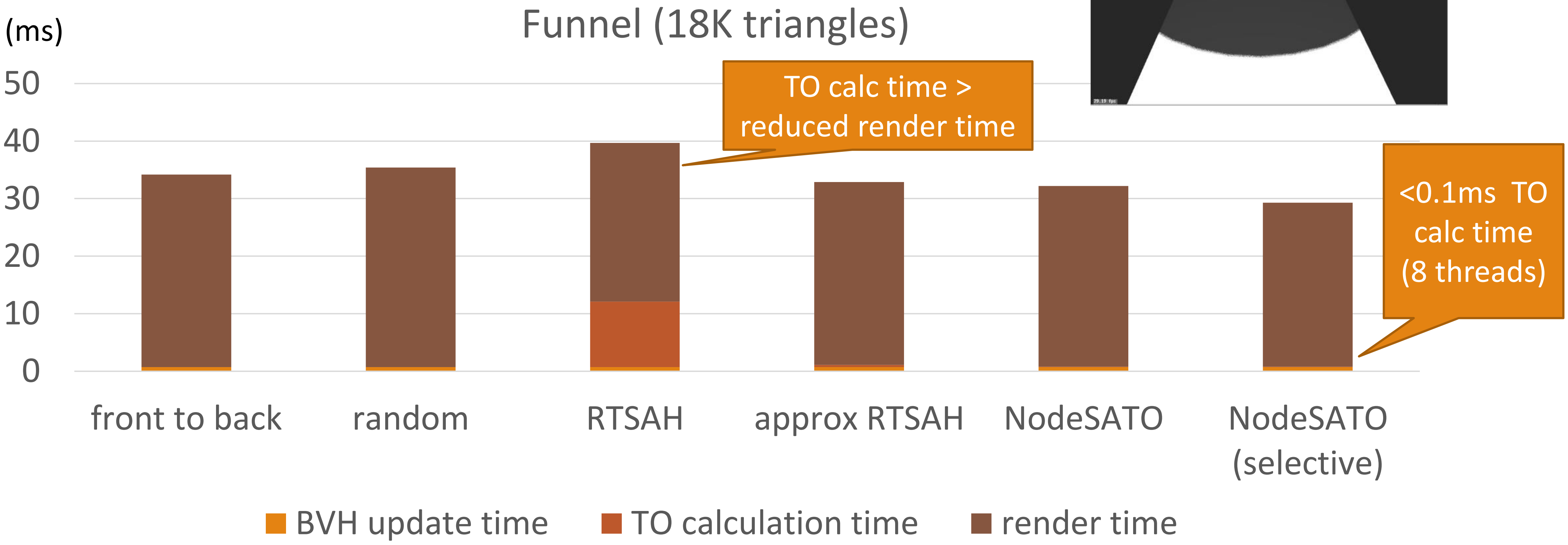


Dynamic Benchmark Setup

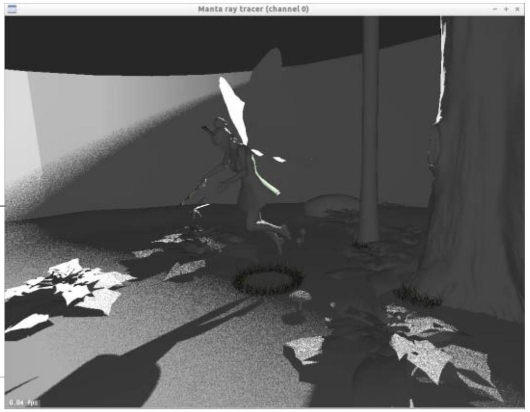
- BVH update algorithm
  - BV refitting [Wald et al. 2007] + tree rotation [Kopta et al. 2012]
- BVH traversal algorithm
  - 8x8 packetized BVH traversal [Wald et al. 2007]
- 1024x768 resolution
- Soft shadows (4 samples per shading point)

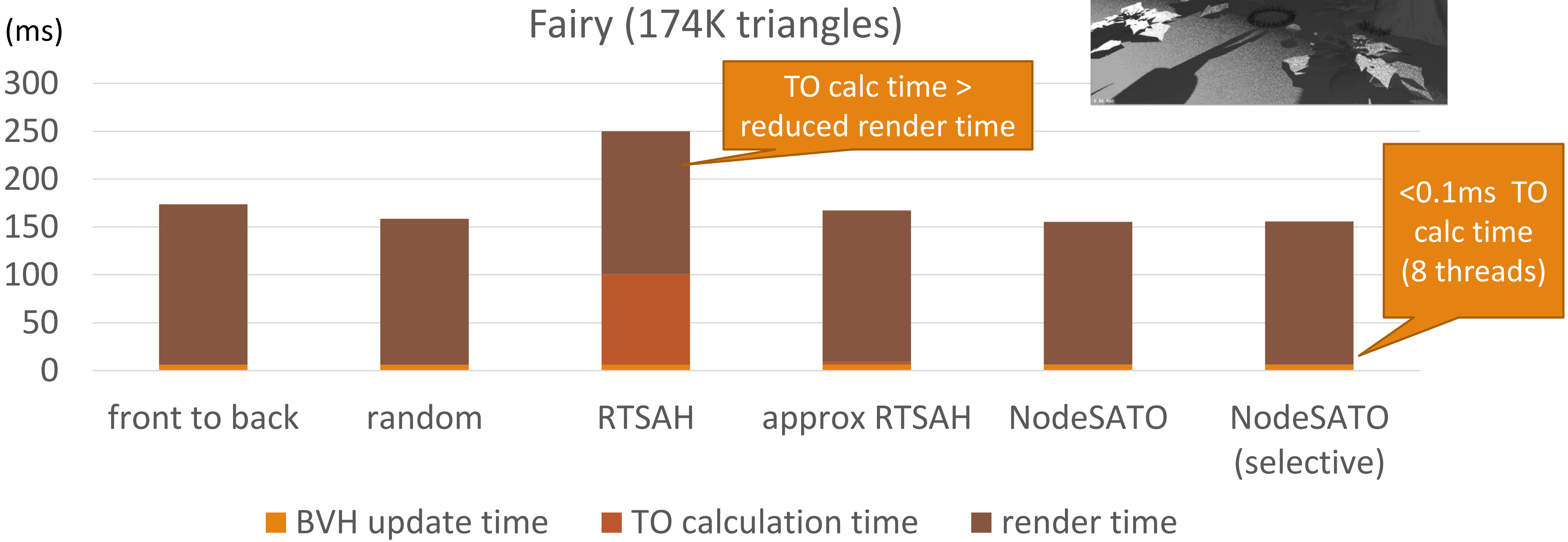
Results in Dynamic Scenes



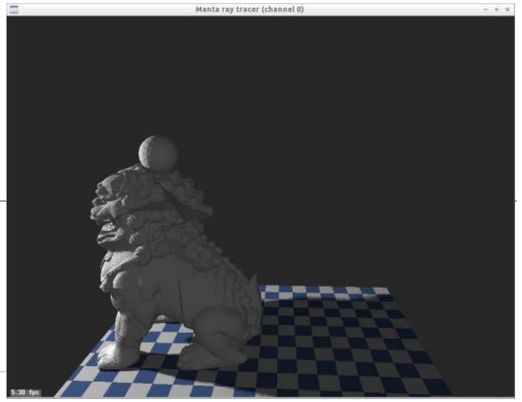


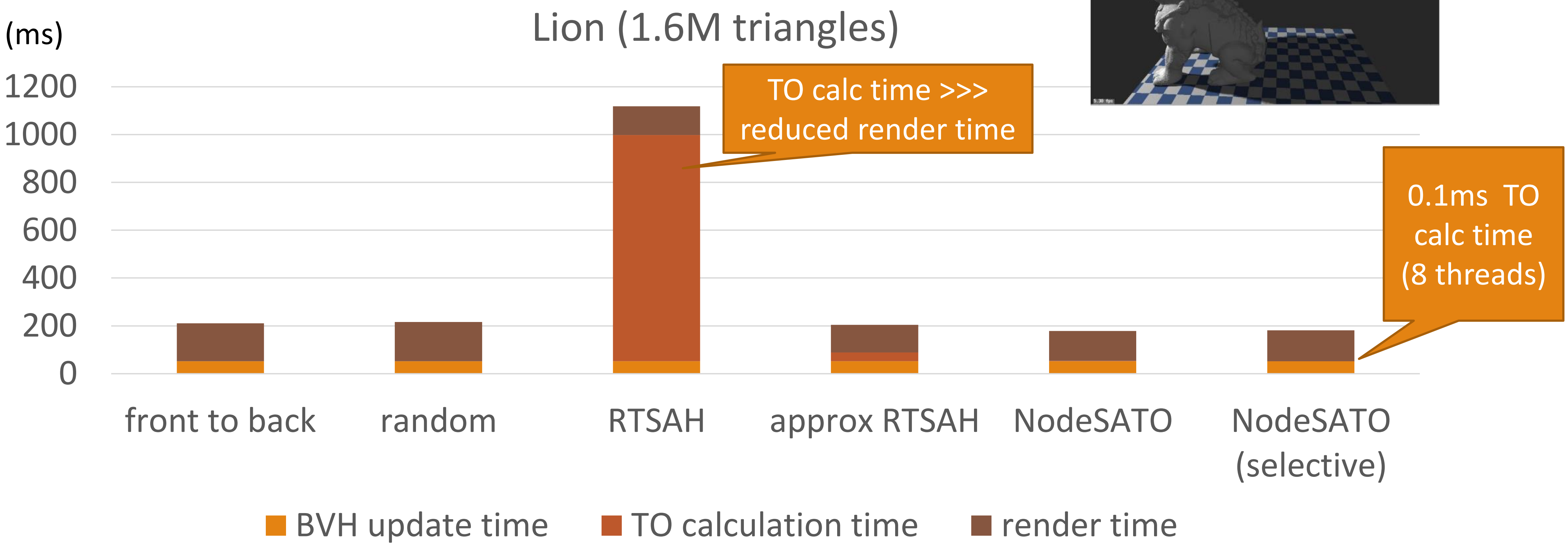
Results in Dynamic Scenes



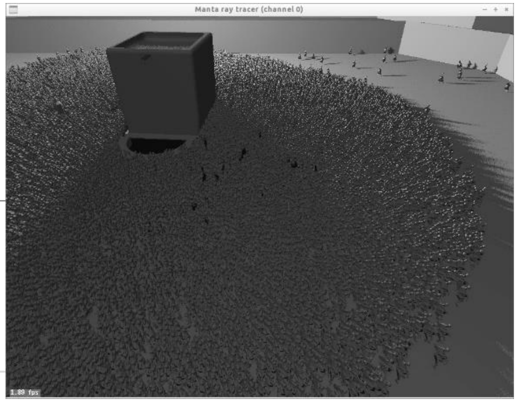


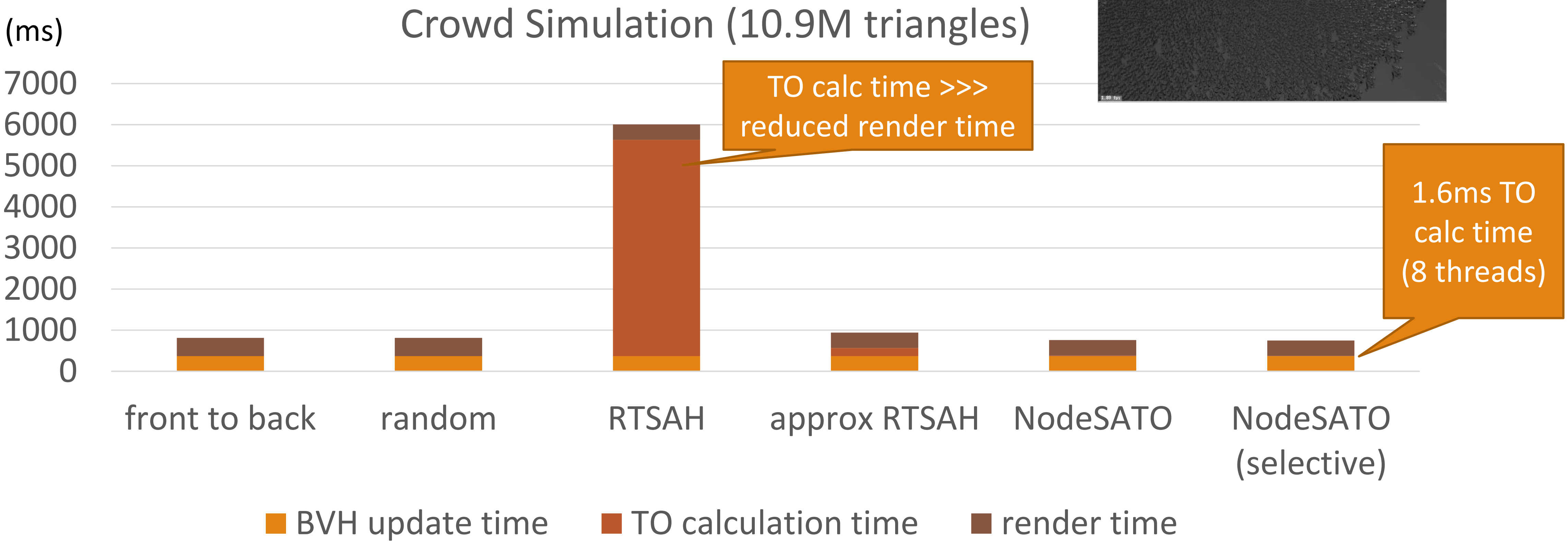
Results in Dynamic Scenes





Results in Dynamic Scenes





Limitations

- SATO does not guarantee performance improvements
  - Only accelerate “occluded shadow” rays
  - Assume SAH-constructed trees (NodeSATO/PrimNumTO)
  - Equal scene primitive sizes → no benefits

Conclusions and Future Work

- NodeSATO: very fast and simple traversal-order heuristic
  - Give traversal priority to the node with larger SA
  - Negligible overheads → very suitable for dynamic scenes
  - Very simple implementation
  - Similar speedup in static scenes compared to RTSAH
- Future work: combination with other methods
  - Static SRDH-constructed tree + dynamic SATO update tree
  - SATO with lazy build [Djeu et al. 2011]

Questions?