Efficient Divide-And-Conquer Ray Tracing using Ray Sampling

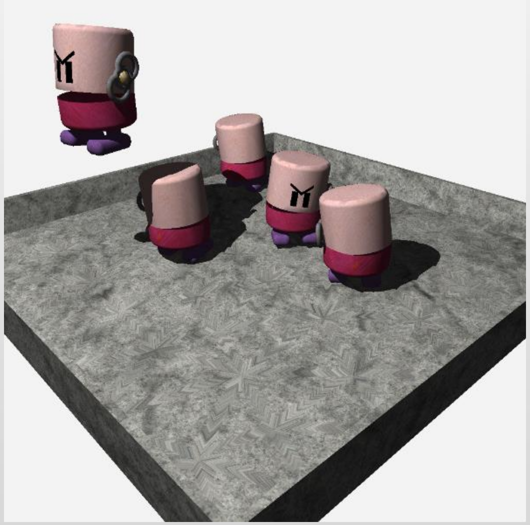
Kosuke Nabata	Wakayama University
Kei Iwasaki	Wakayama University/UEI Research
Yoshinori Dobashi	Hokkaido University/JST CREST
Tomoyuki Nishita	UEI Research/Hiroshima Shudo University

Outline

- Introduction
- Previous Work
  - Divide-And-Conquer Ray Tracing
- Proposed Method
- Results
- Conclusions and Future Work

Introduction

- Recent advances in ray tracing
  - construct acceleration data structures *before* ray tracing
  - grid, kd-tree, bounding volume hierarchy (BVH)
  - acceleration data structures require extensive memory
  - required memory is not determined before construction



[Wald 2006]

[Zhou 2008]

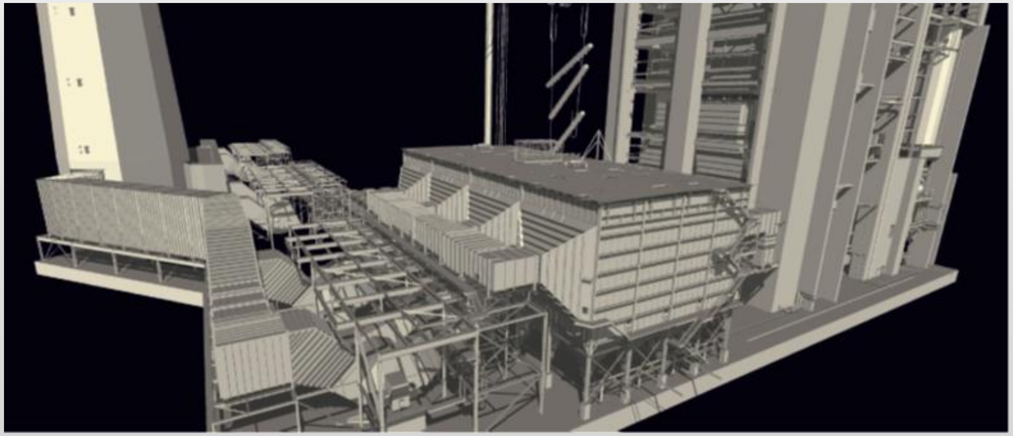
[Wald 2007]

Divide-And-Conquer Ray Tracing (DACRT)

- **Ray tracing based on divide-and-conquer algorithm**

  [Keller et al. 2011] [Mora 2011] [Afra 2012]

  - **trace rays and construct acceleration data structures *simultaneously***
  - **no storage cost for acceleration data structures**
  - **required memory is minimal and deterministic**

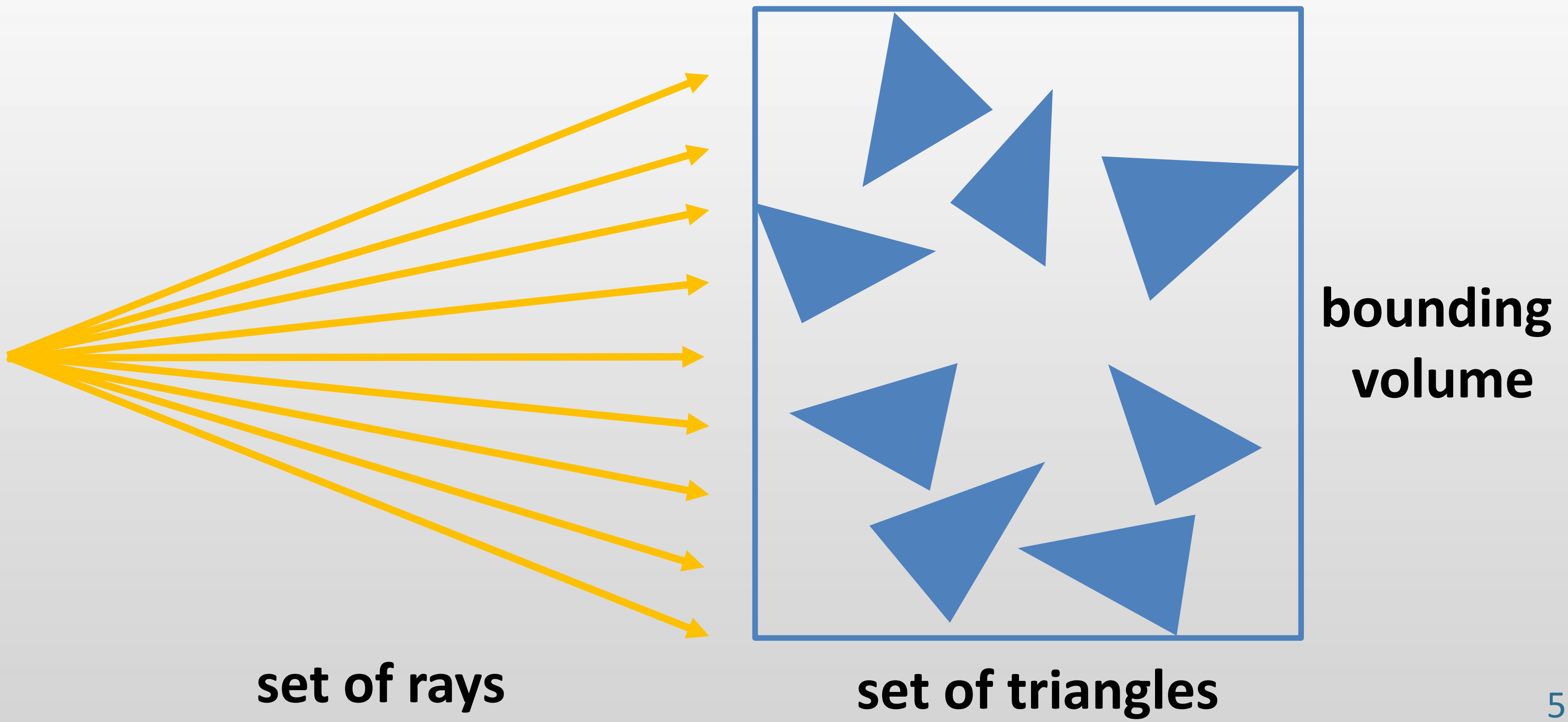


[Mora 2011]

[Afra 2012]

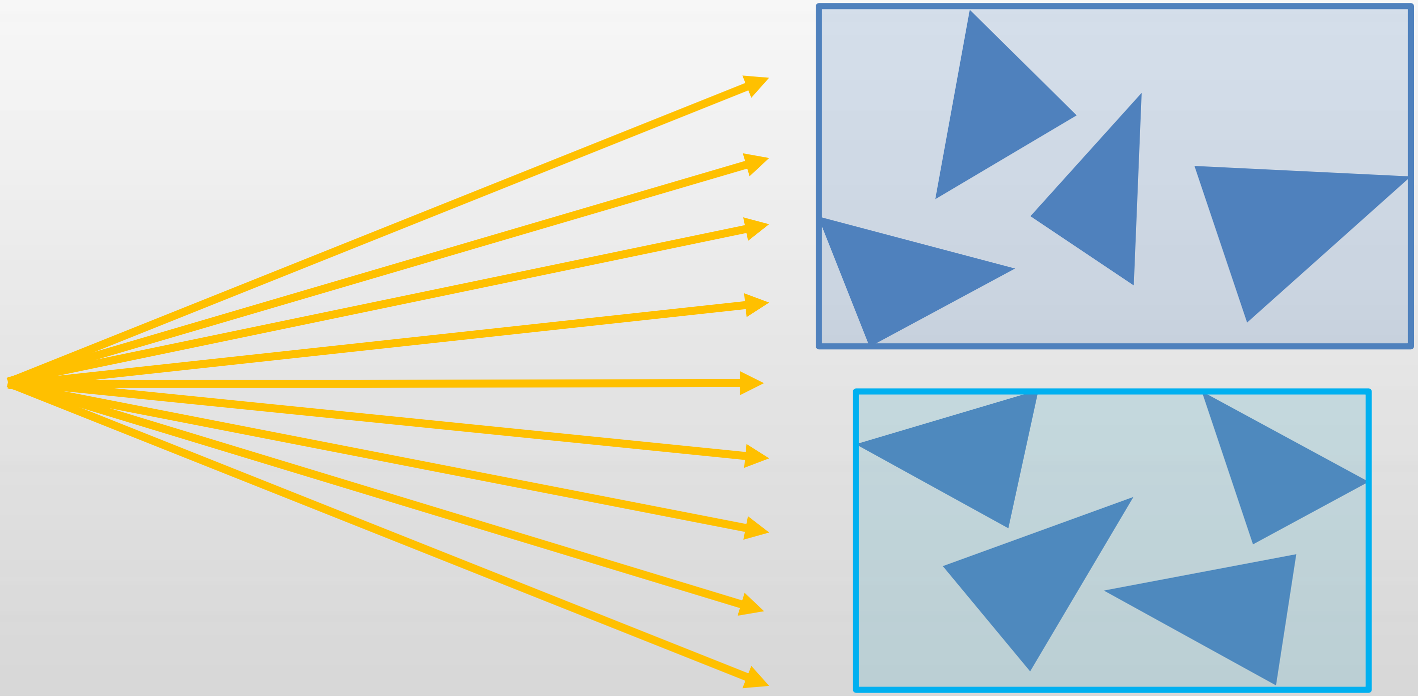
Divide-And-Conquer Ray Tracing (DACRT)

- **Solve intersection problem between rays and primitives using divide-and-conquer algorithm**
  - **triangles are used as primitives**



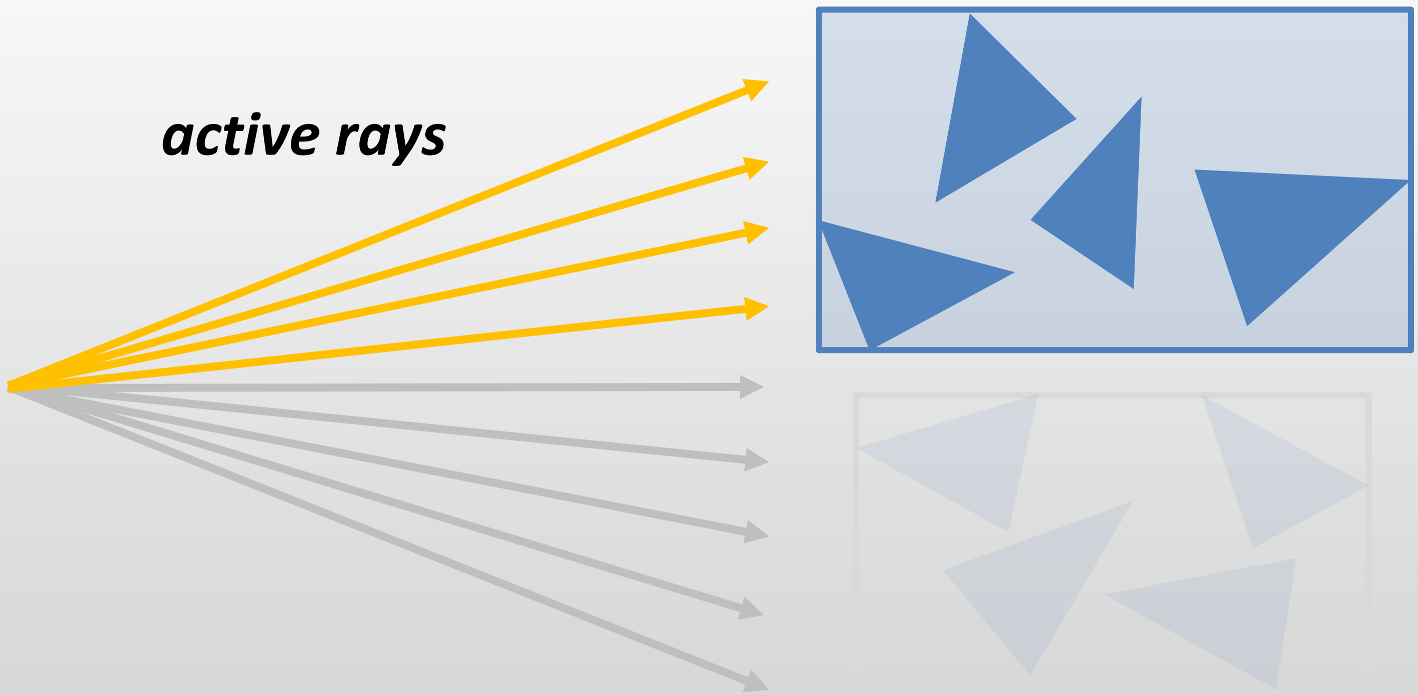
Divide-And-Conquer Ray Tracing (DACRT)

- **Partition a set of triangles into subsets of triangles**
  - **space partitioning (kd-tree)**
  - ***object partitioning (BVH)***



Divide-And-Conquer Ray Tracing (DACRT)

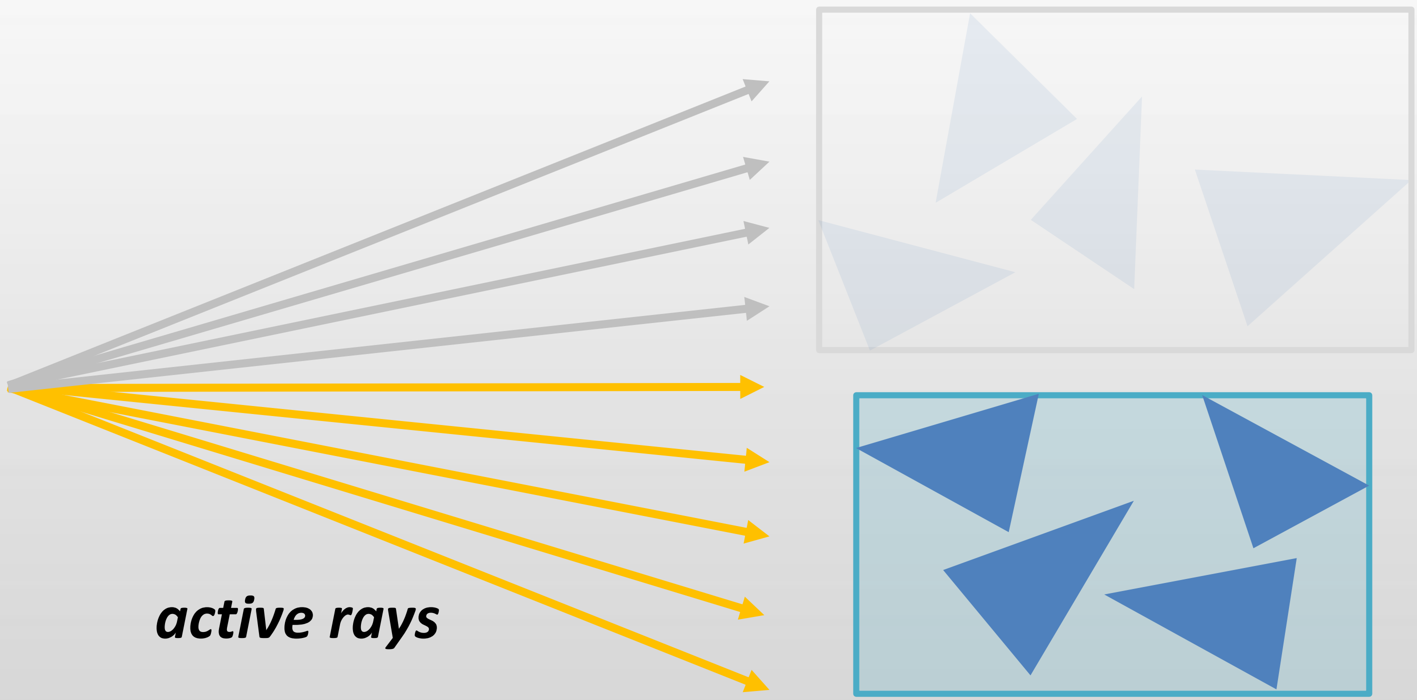
- **Partition a set of rays intersecting bounding volume**



ray filtering

Divide-And-Conquer Ray Tracing (DACRT)

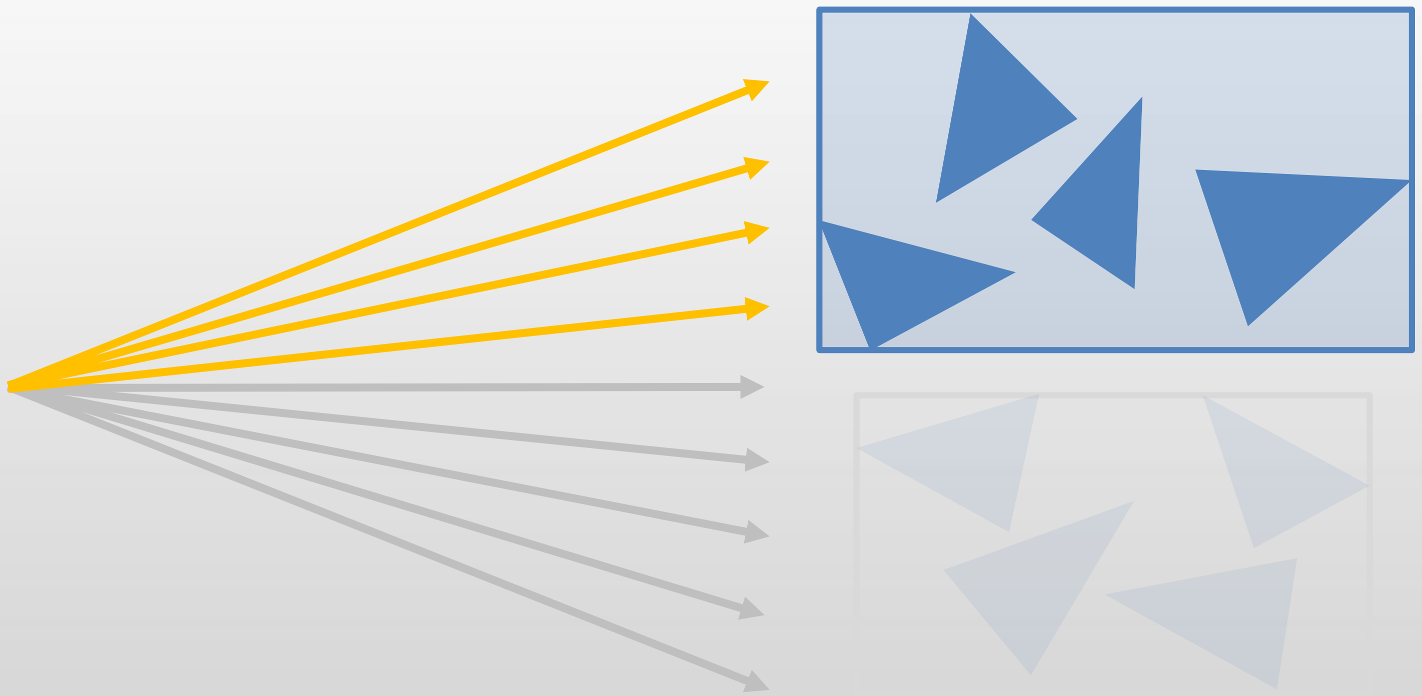
- **Partition a set of rays intersecting bounding volume**



ray filtering

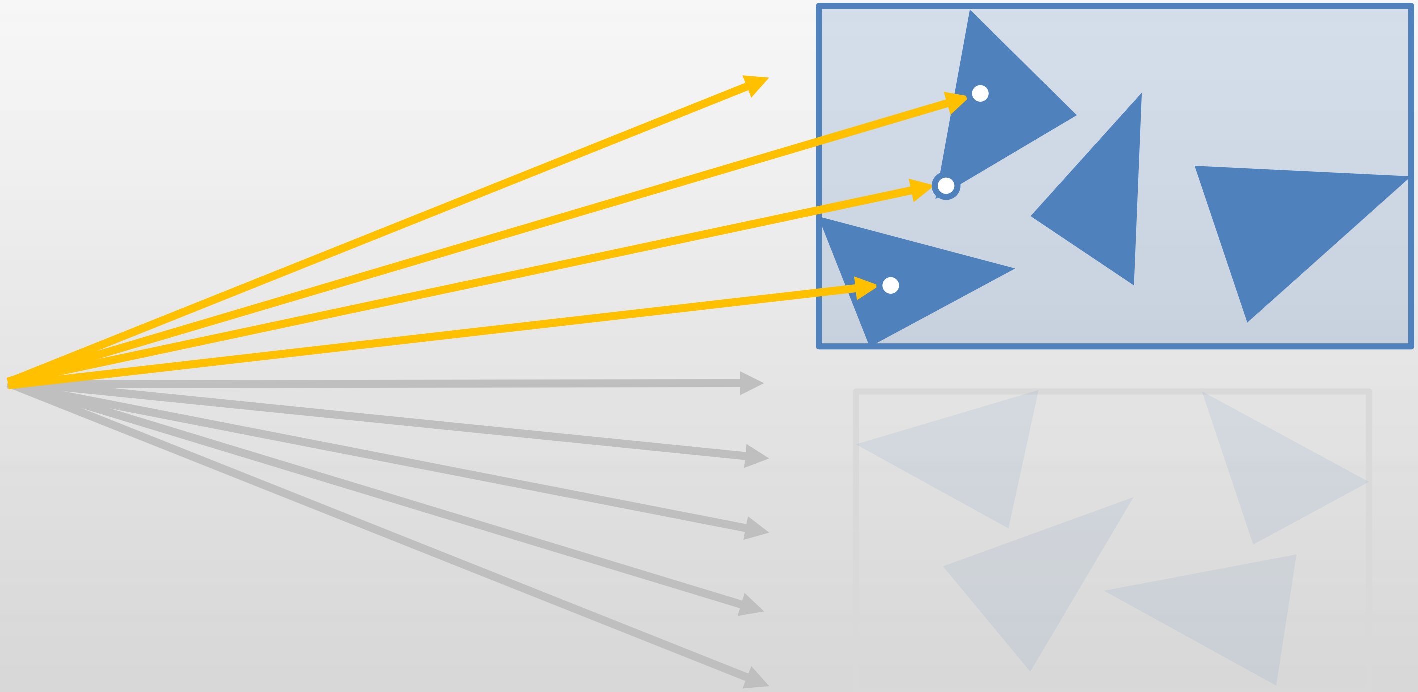
Divide-And-Conquer Ray Tracing (DACRT)

- **Solve intersection problem directly**
  - **if numbers of rays or triangles are sufficiently small**



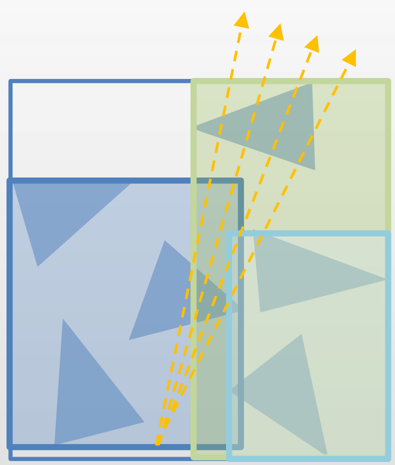
Divide-And-Conquer Ray Tracing (DACRT)

- **Solve intersection problem directly**
  - **if numbers of rays or triangles are sufficiently small**

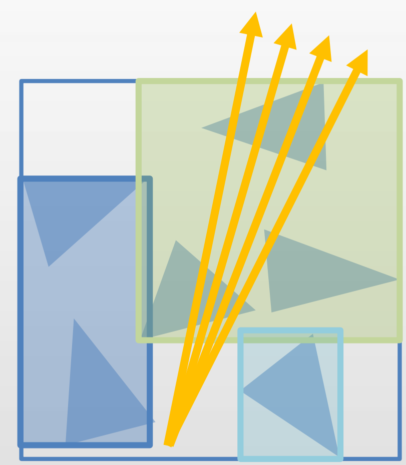


Problems of Previous DACRT Methods

- **Subdivide problems based on triangle distribution only**
  - **partition triangles assuming *uniform* distribution of rays**
  - **inefficient for concentrated distribution of rays**



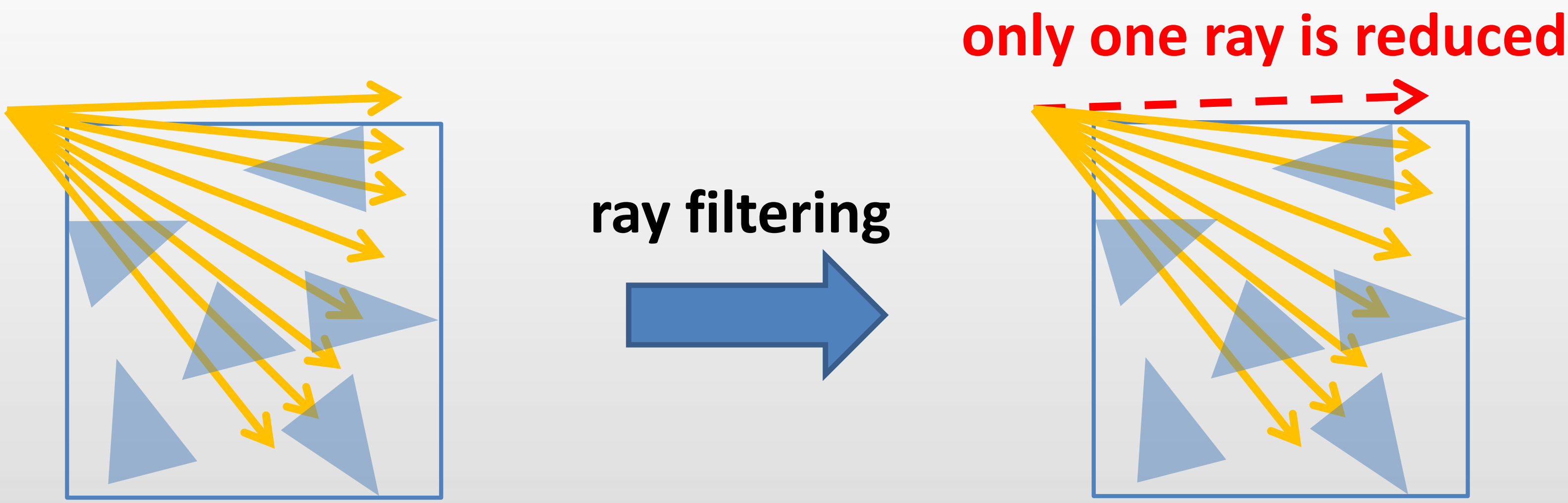
partitioning based on distribution of triangles only



partitioning based on distributions of *triangles and rays*

Problems of Previous DACRT Methods

- **Ray filtering may not reduce number of active rays**
  - **require many ray/bounding volume intersection tests**
  - **ray filtering is computationally expensive**



inefficient case of ray filtering

Contributions of Our DACRT Method

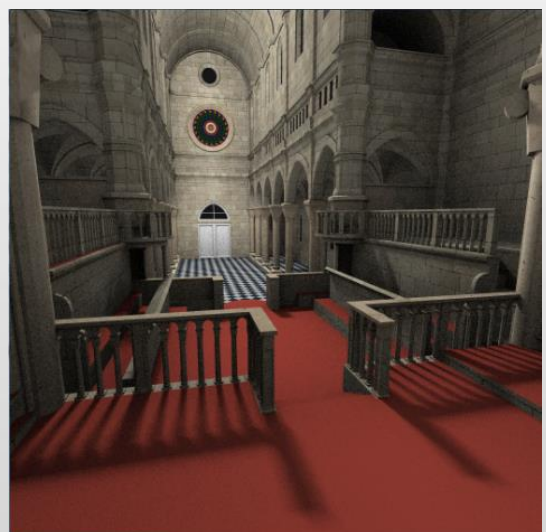
- **Accelerate ray tracing using ray sampling**
  - **efficient partitioning and ray traversal**
- **Derive a new cost metric to avoid inefficient ray filtering**
  - **simple but efficient**

rendering result of our method

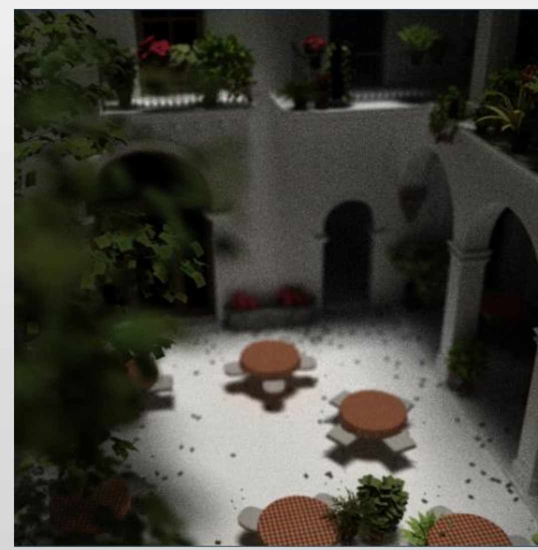
Features of Our DACRT Method

- **Accelerate tracing of many types of rays by a factor of 2**
  - **primary rays, secondary rays, random rays**
  - **reflection/refraction, ambient occlusion, path tracing**
- **Performance gain increases as number of rays increases**
  - **beneficial for high resolution images and anti-aliasing**

area light, specular reflection



ambient occlusion (AO)



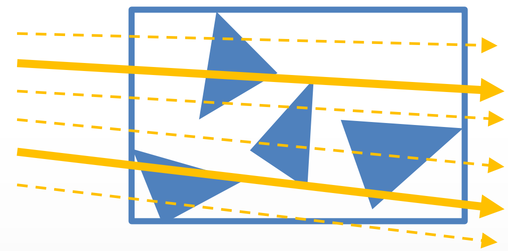
path tracing, depth of field

Outline

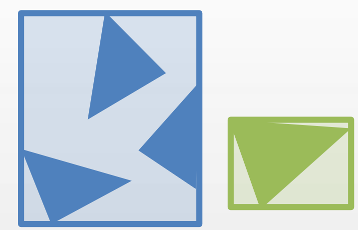
- Introduction
- Previous Work
  - Divide-And-Conquer Ray Tracing
- **Proposed Method**
- Results
- Conclusions and Future Work

Overview of Our Method

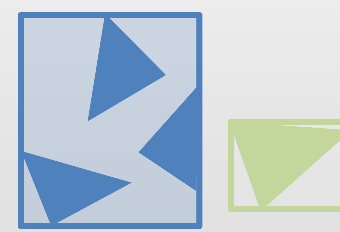
Ray Sampling



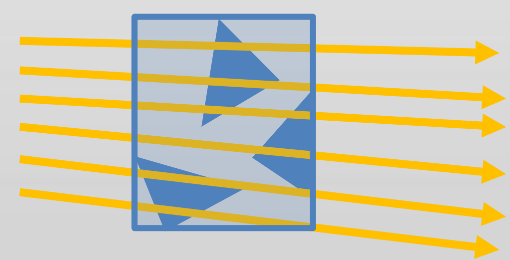
Partitioning using Cost Function



Determining Traversal Order

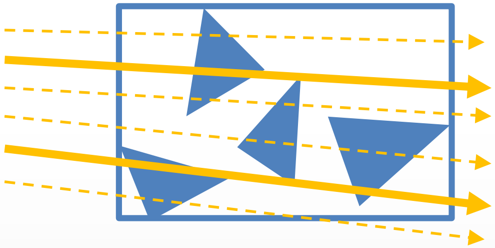


Traversal with Skip Ray Filtering



Overview of Our Method

Ray Sampling



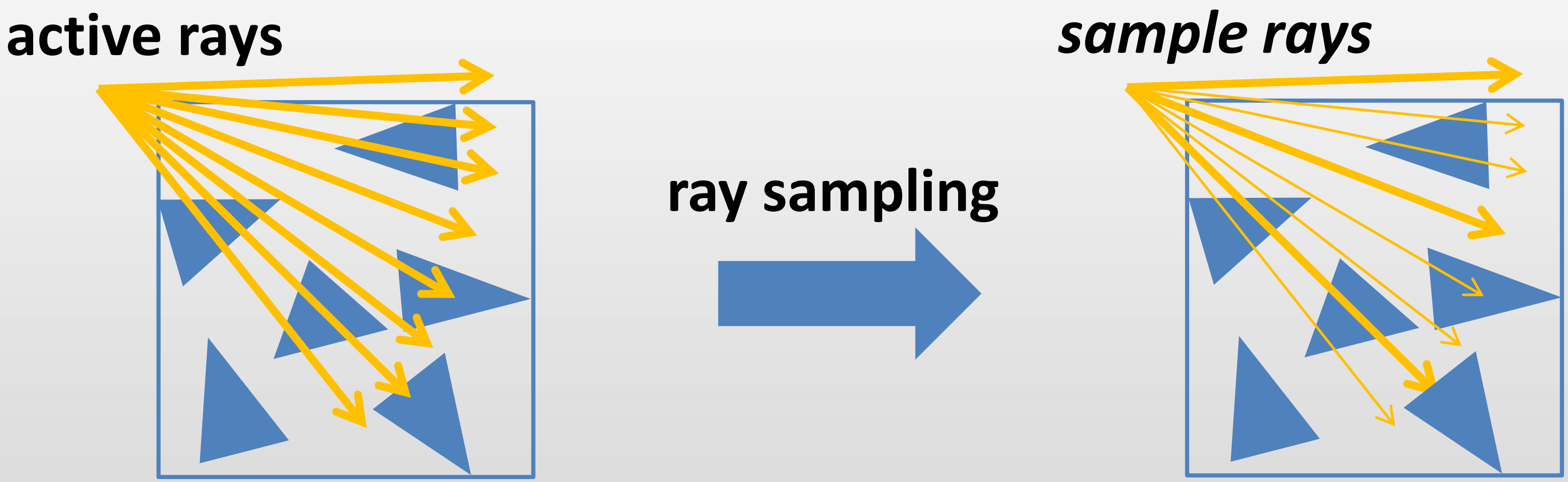
Partitioning using Cost Function

Determining Traversal Order

Traversal with Skip Ray Filtering

Ray Sampling

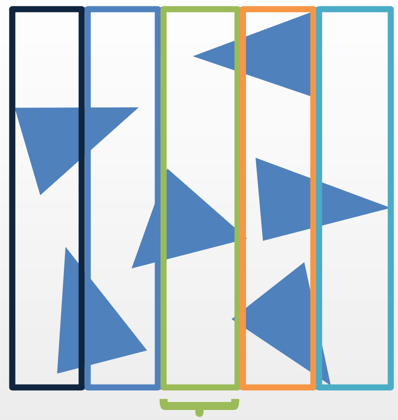
- **Trace a small subset of active rays : *sample rays***
  - **ray sampling is performed if number of active rays is sufficiently large**



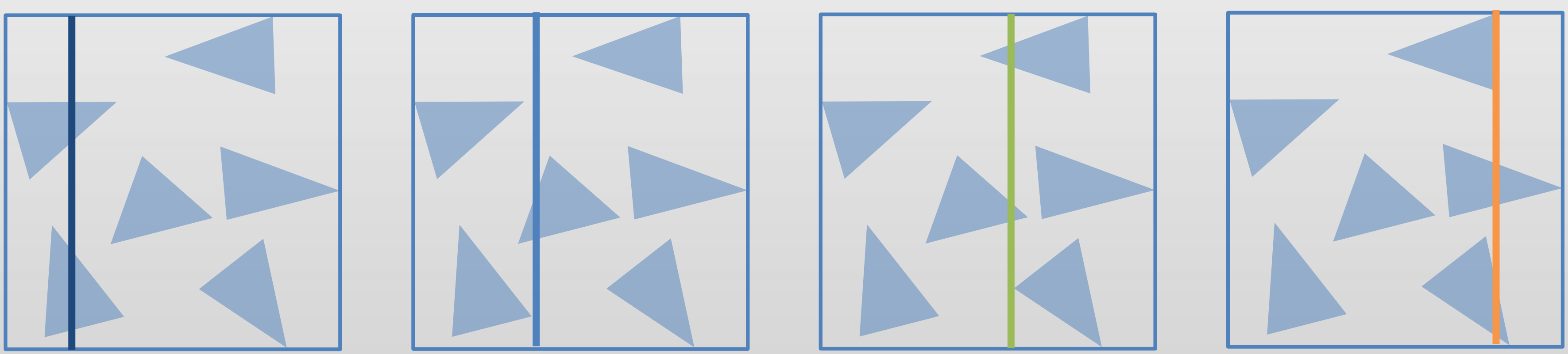
Ray Sampling

- **Subdivide bounding volume into bins**

[Wald 2007]



bin

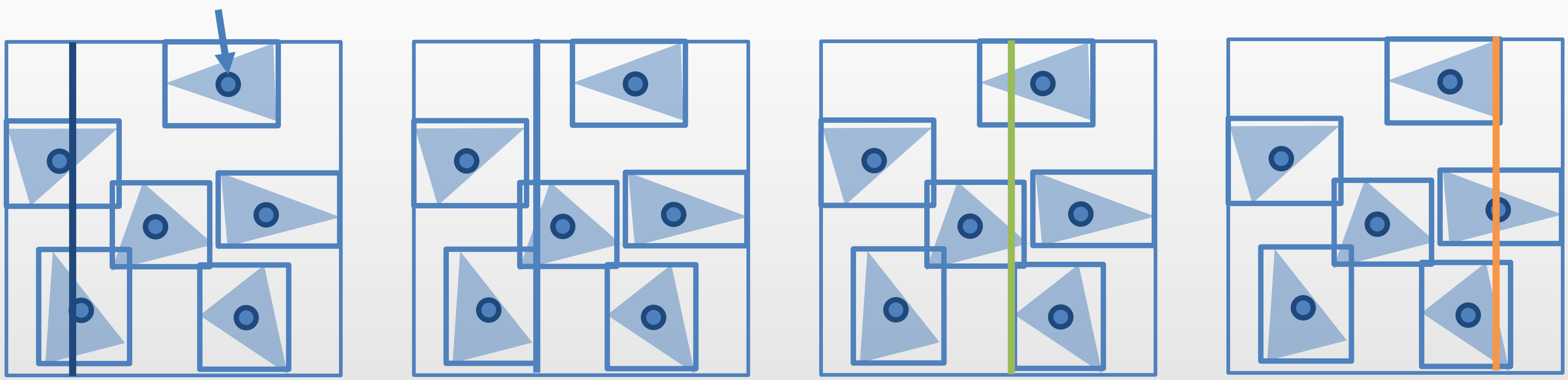


partitioning candidates

Ray Sampling

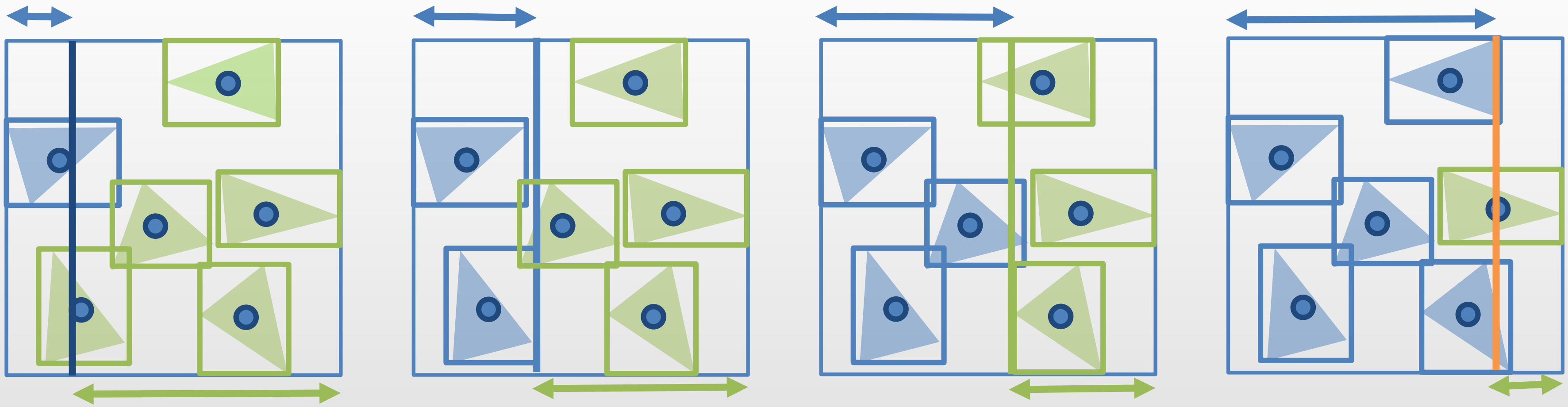
- **Calculate center of triangle’s axis-aligned bounding box**

[Wald 2007]



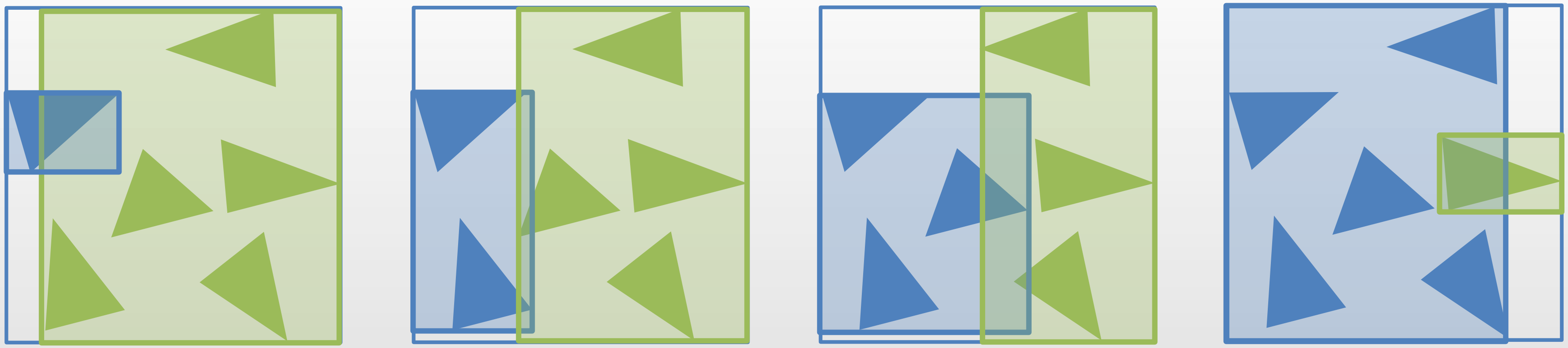
Ray Sampling

- **Partition set of triangles into two disjoint subsets**



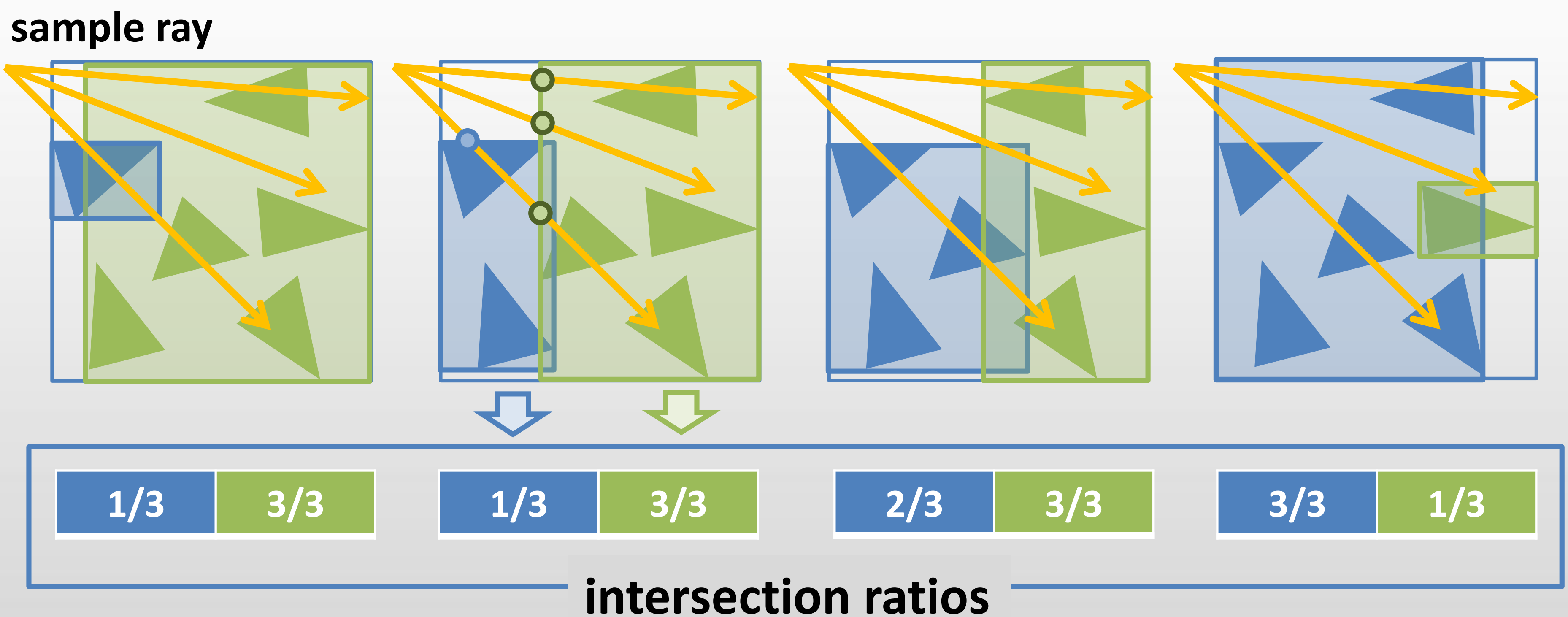
Ray Sampling

- **Partition set of triangles into two disjoint subsets**



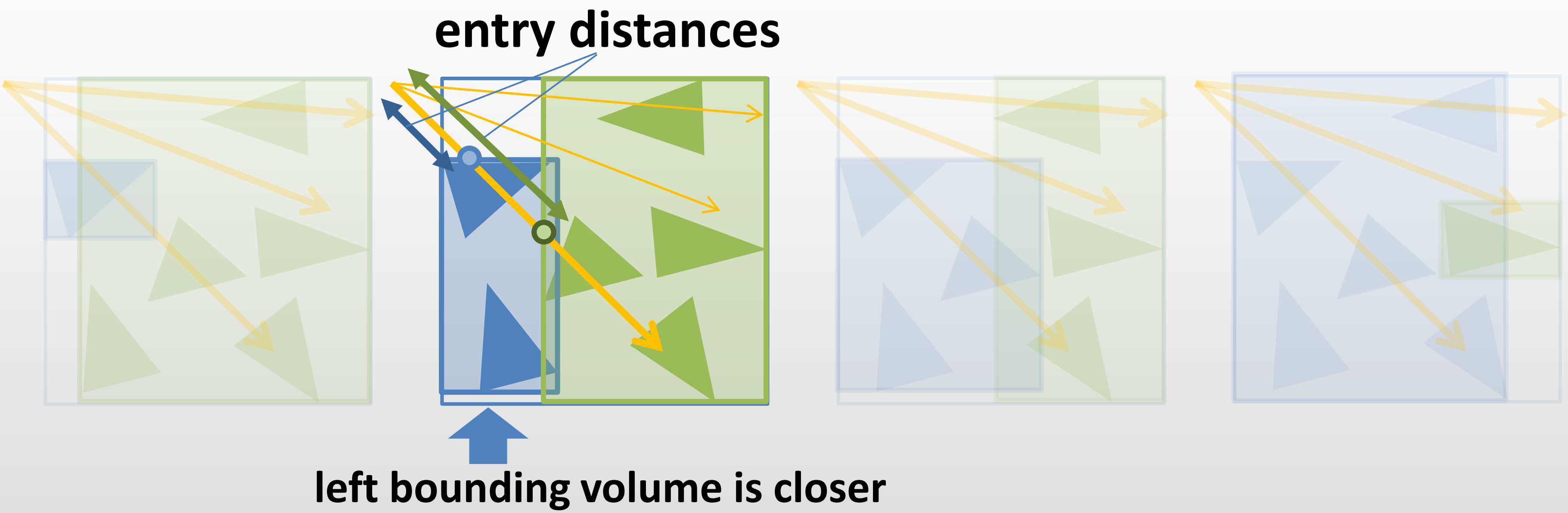
Ray Sampling

- Calculate *intersection ratio* α for each bounding volume
  - ratio of sample rays intersecting each bounding volume



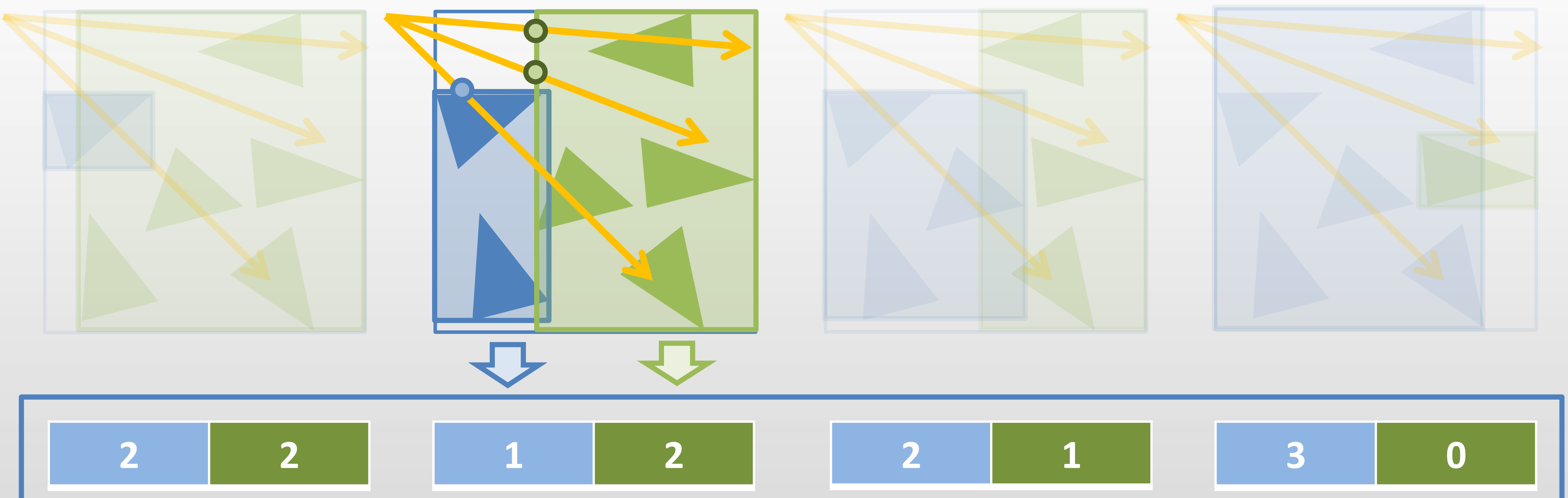
Ray Sampling

- Calculate *entry distance* for each bounding volume
  - distance from ray origin to nearest intersection point



Ray Sampling

- **Count closer sample rays for each bounding volume**
  - **number of sample rays with smaller entry distances**



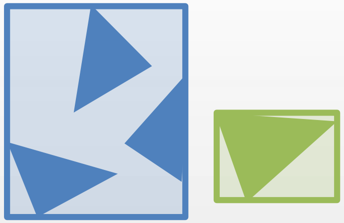
number of closer sample rays

Overview of Our Method

Ray Sampling

Partitioning using Cost Function

Determining Traversal Order

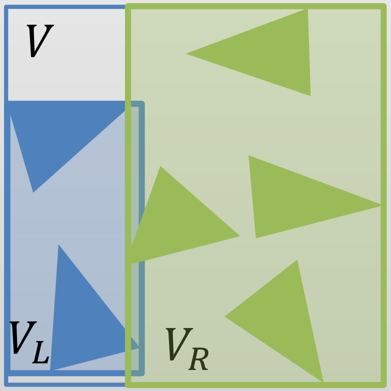


Partitioning using Cost Function

- **Minimize cost function for efficient partitioning**

$$C(V \to \{V_L, V_R\}) = \underbrace{\boxed{C_T + C_I}}_{\text{constant}}(p_L N_L + p_R N_R)$$

$$C(V \to \{V_L, V_R\}) = p_L N_L + p_R N_R$$



V, V_L, V_R	bounding volumes
C_T, C_I	costs of ray/BV, ray/triangle intersections
N_L, N_R	numbers of triangles in V_L, V_R
p_L, p_R	probabilities of rays intersecting V_L, V_R

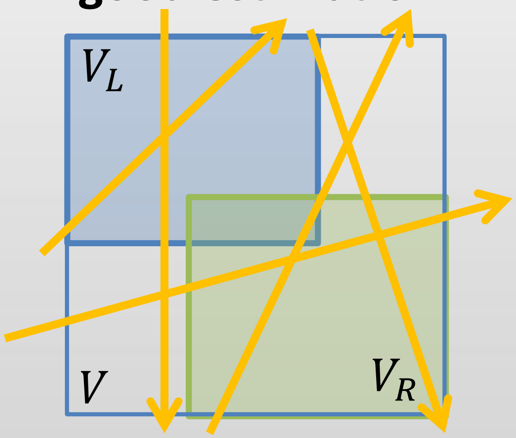
Cost Function of Previous DACRT Method

- **Surface Area Heuristic (SAH) approximates probabilities with ratios of surface areas**

$$C(V \rightarrow \{V_L, V_R\}) = \boxed{\frac{SA(V_L)}{SA(V)}} N_L + \boxed{\frac{SA(V_R)}{SA(V)}} N_R$$

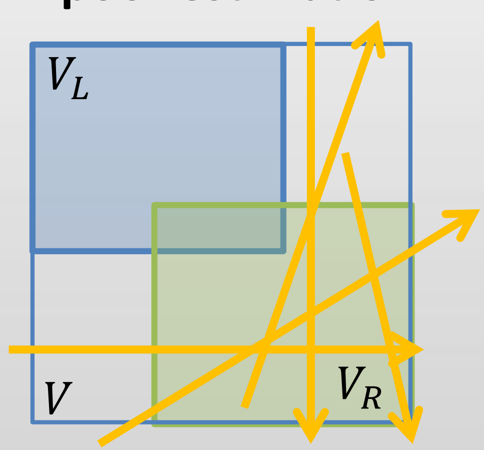
$SA(V)$	**surface area of bounding volume** V

SAH provides good estimation

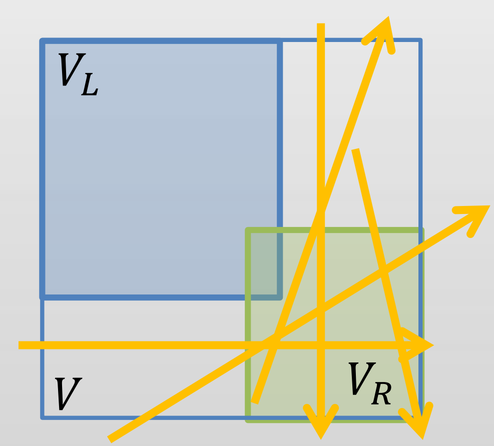


uniform distribution

SAH provides poor estimation



non-uniform distribution



our method

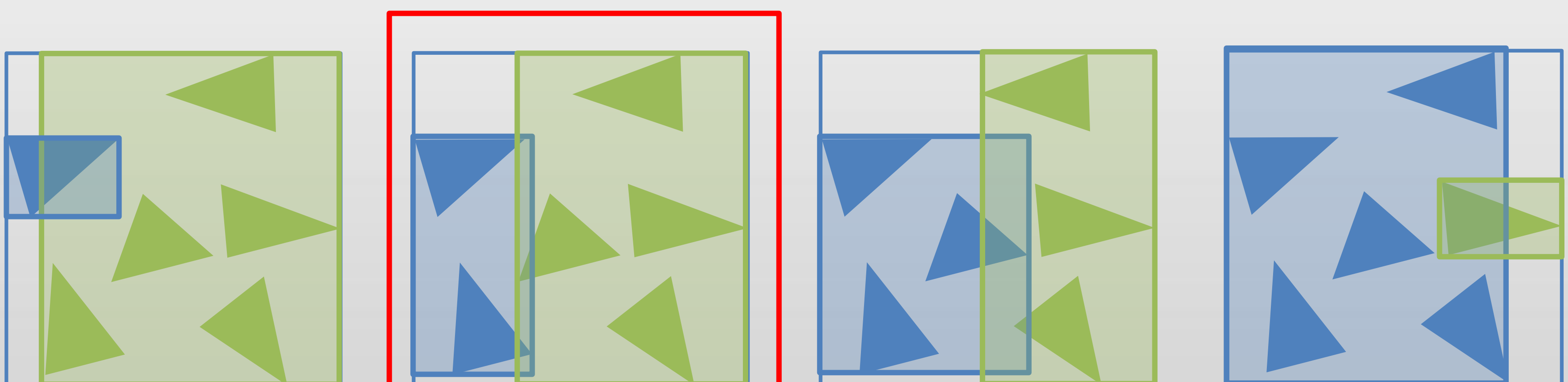
Partitioning using Cost Function

- **Estimate probabilities of ray hit using intersection ratios**
  - **use *actual* distribution of rays for partitioning**

$$C(V \to \{V_L, V_R\}) = \boxed{\alpha_L}\, N_L + \boxed{\alpha_R}\, N_R$$

1/3	3/3	1/3	3/3	2/3	3/3	3/3	1/3

intersection ratios



$C = 16/3$ $C = 14/3$ $C = 15/3$ $C = 16/3$

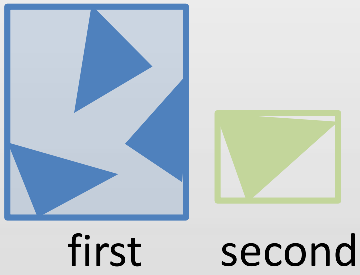
Overview of Our Method

Ray Sampling

Partitioning using Cost Function

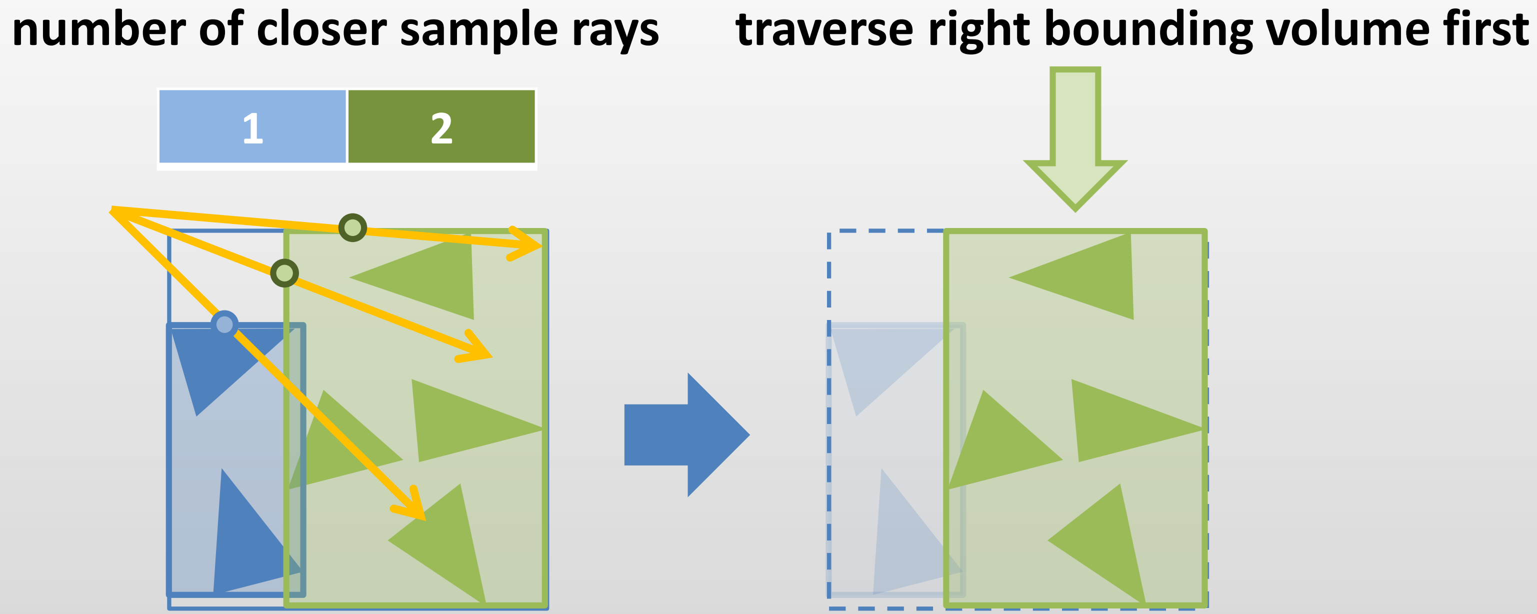
Determining Traversal Order

Traversal with Skip Ray Filtering



Traversal Order Determination

- **Traverse bounding volume with larger number of closer sample rays first**
  - **additional operation is only a comparison**



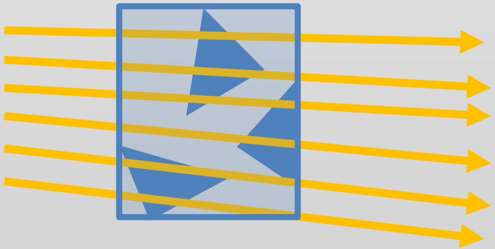
Overview of Our Method

Ray Sampling

Partitioning using Cost Function

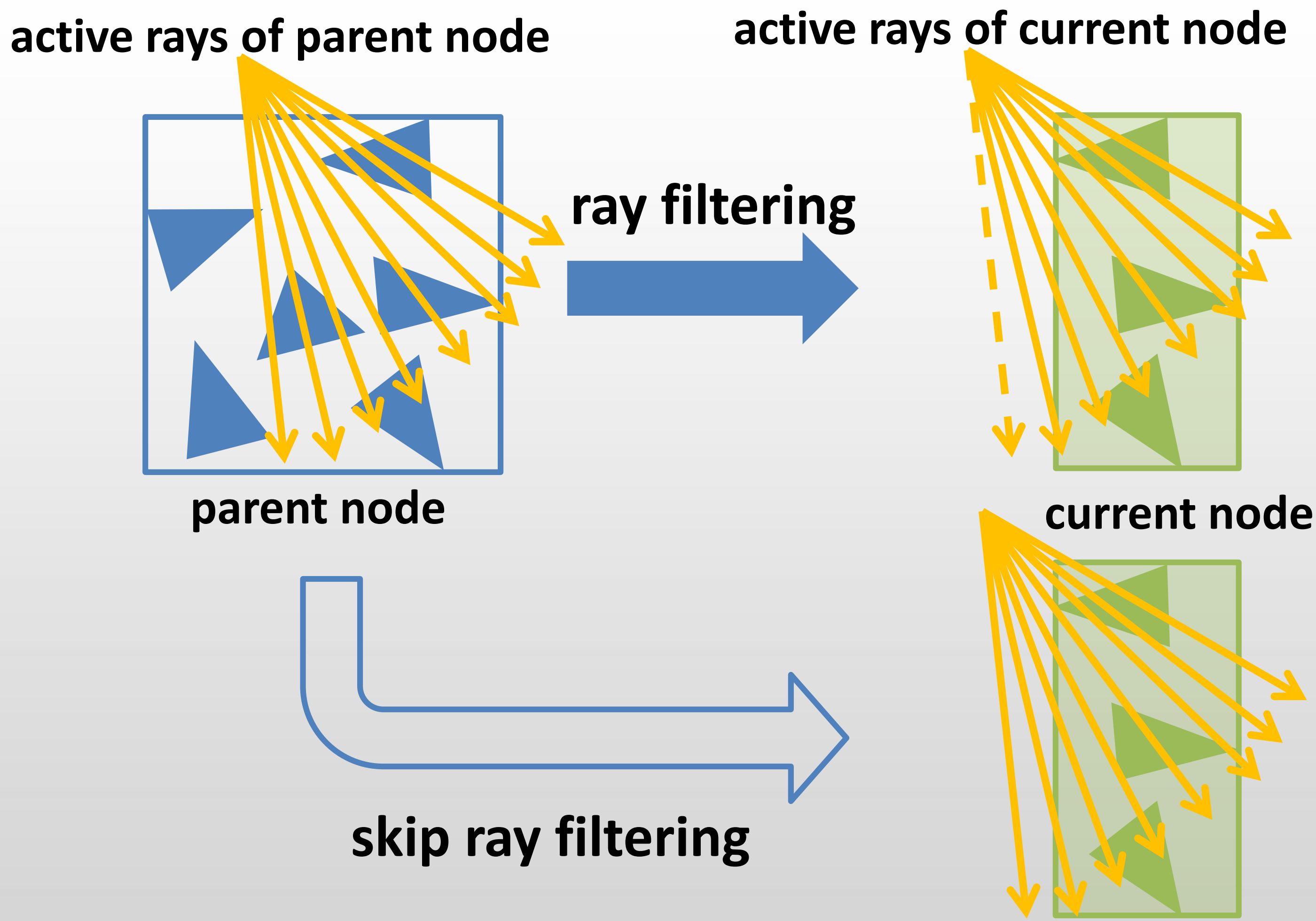
Determining Traversal Order

Traversal with Skip Ray Filtering



Inefficient Case of Ray Filtering

- **Most of active rays intersect bounding volume**

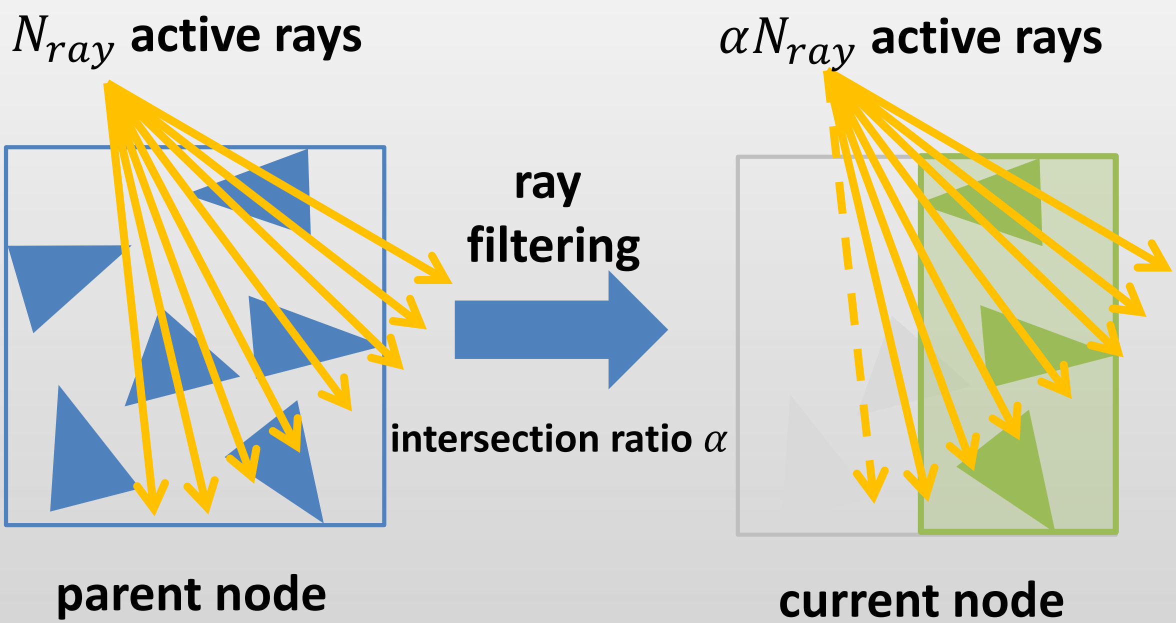


Cost Metric for Ray Filtering

- **Cost C_{int} for ray filtering**

$$C_{int} = N_{ray} C_{bv}$$

C_{bv} **:ray/BV intersection test cost**

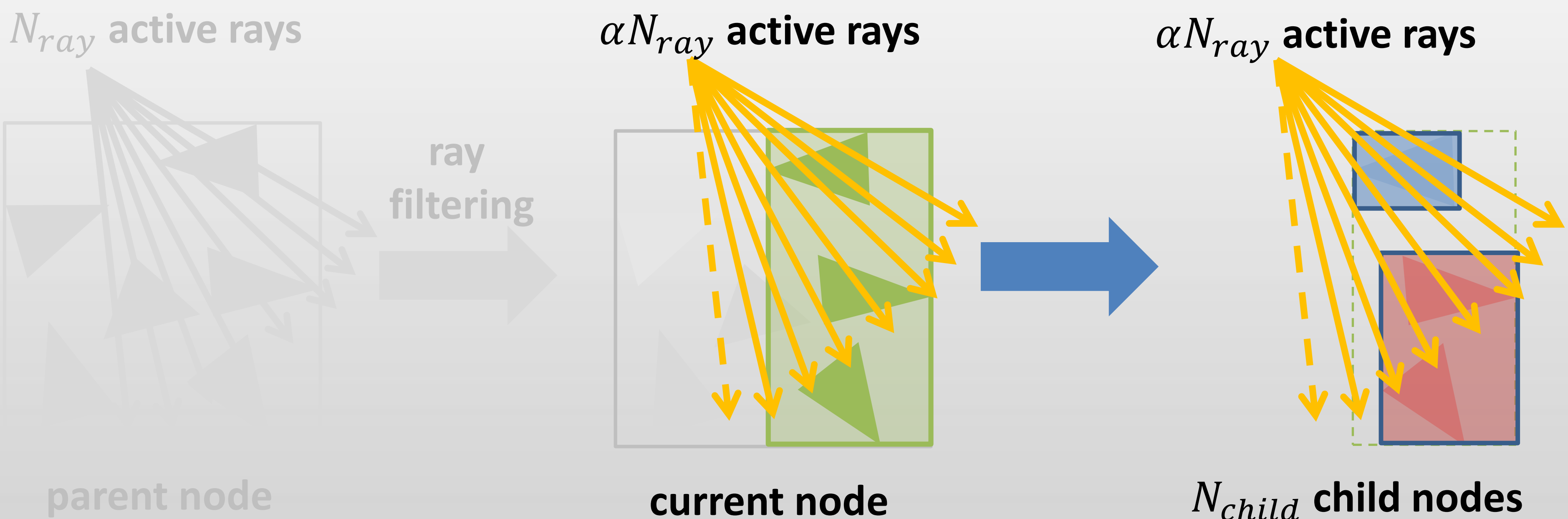


Cost Metric for Ray Filtering

- **Cost C_{int} for ray filtering**

$$C_{int} = N_{ray} C_{bv} \qquad + \quad \alpha N_{ray} C_{child} N_{child}$$

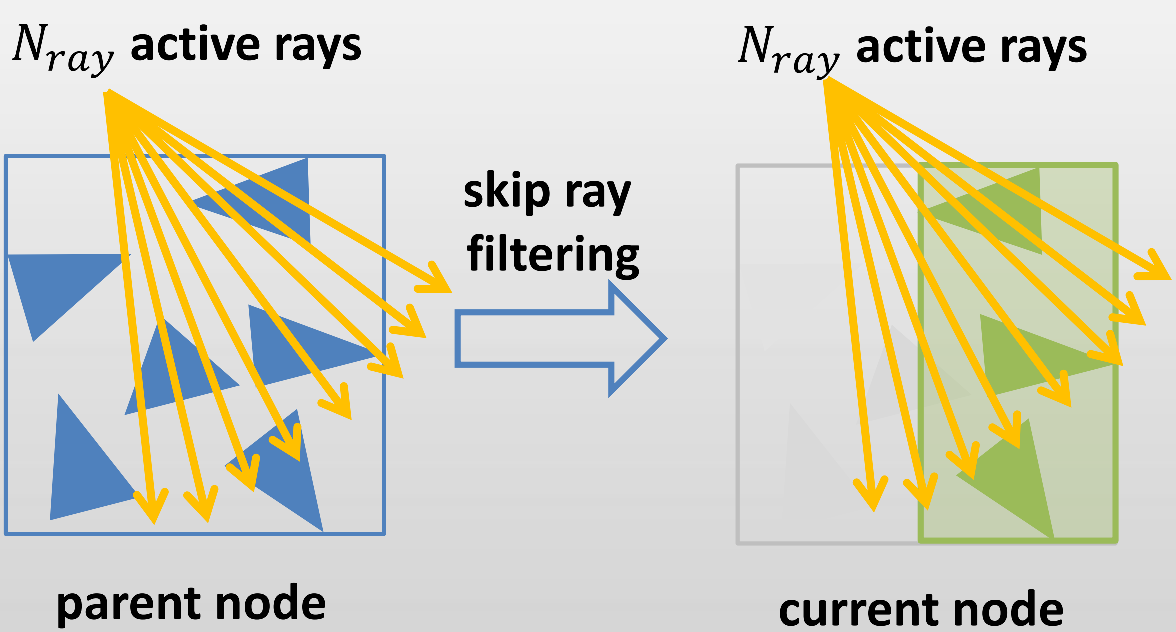
C_{child} **:child node/ray intersection test cost**



Cost Metric for Skip Ray Filtering

- **Cost C_{skip} for skip ray filtering**

$$C_{skip} = \quad 0$$

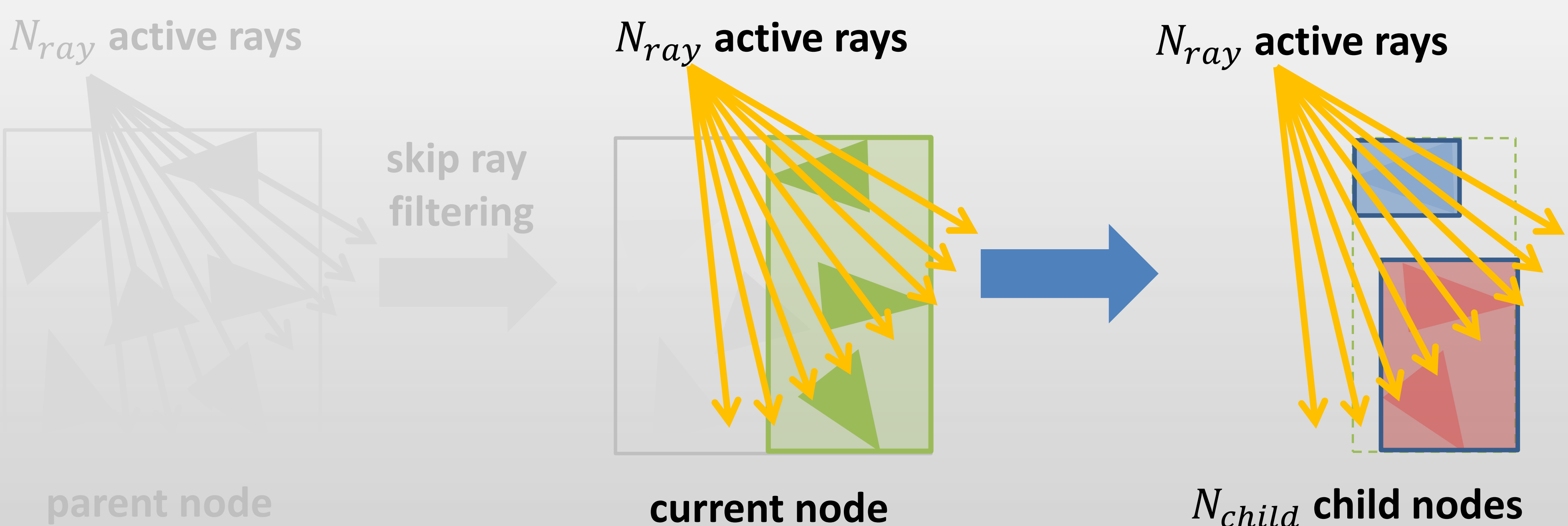


Cost Metric for Skip Ray Filtering

- **Cost C_{skip} for skip ray filtering**

$$C_{skip} = 0 + N_{ray} C_{child} N_{child}$$

C_{child} **:child node/ray intersection test cost**



Determine Skip Ray Filtering

- **Skip ray filtering if C_{int} > C_{skip}**

- **Skipping criterion for intersection ratio α**

$$\alpha > 1 - \frac{C_{bv}}{N_{child} C_{child}}$$

- **Skipping criterion for a non-leaf node of binary BVH**

$$\alpha > 0.5$$

Outline

- Introduction
- Previous Work
  - Divide-And-Conquer Ray Tracing
- Proposed Method
- **Results**
- **Conclusions and Future Work**

Computational conditions

- CPU : Intel Core i7 2.67GHz
- Computational times of ray tracing
  - single thread with SSE
  - 4096^2 image (rendered as 512^2 with 64 MSAA)
  - ray generation, shading are not included
- Comparison with Afra’s method
  - SAH cost function/with ray filtering
- Comparison with Mora’s method

Results (1/3)

- **Our method accelerates ray tracing by a factor of 2**
  - **primary rays·secondary rays**

Sibenik (75K tri.)	
point light / specular reflection	area light / specular reflection
1.86x (27.3s/<u>14.7s</u>)	**1.94x** (22.3s/<u>11.5s</u>)

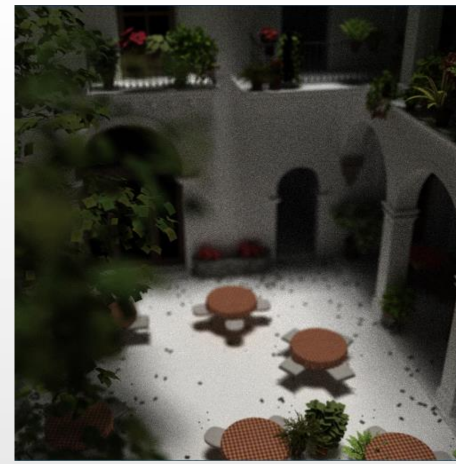
Results (2/3)

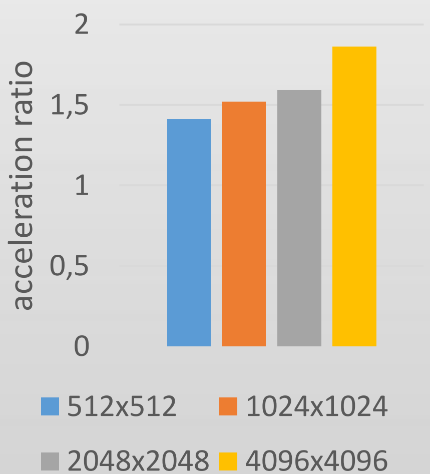
- **Our method accelerates ray tracing by a factor of 2**
  - **primary rays·secondary rays·random rays**

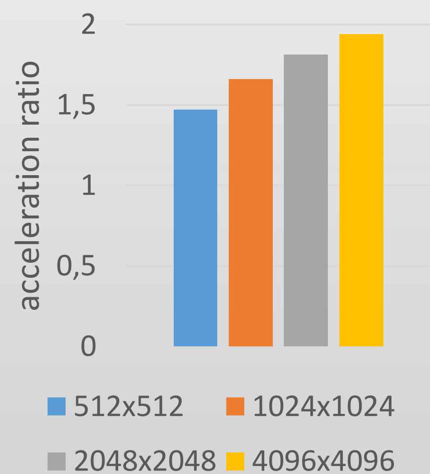
Sponza (262K tri.)	San Miguel (3.3M tri.)
path tracing	**path tracing/depth of field**
1.39x (136s/98s)	**1.25x** (216s/173s)

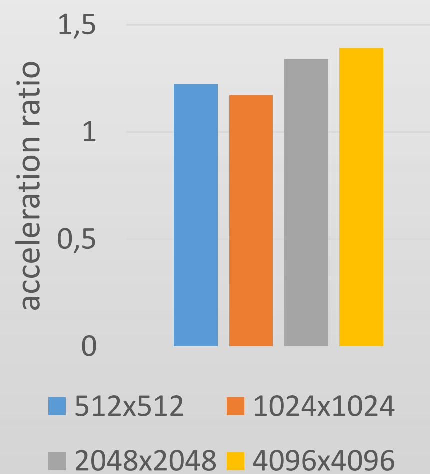
Results (3/3)

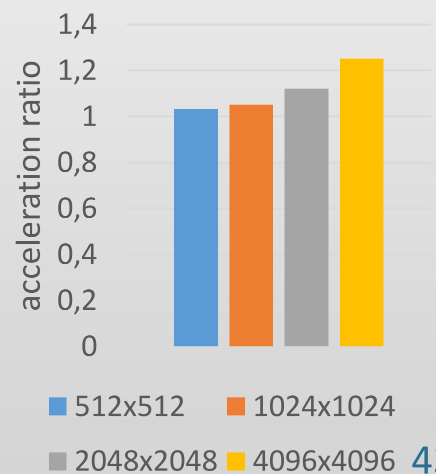
- **Acceleration ratio increases for high resolution images**











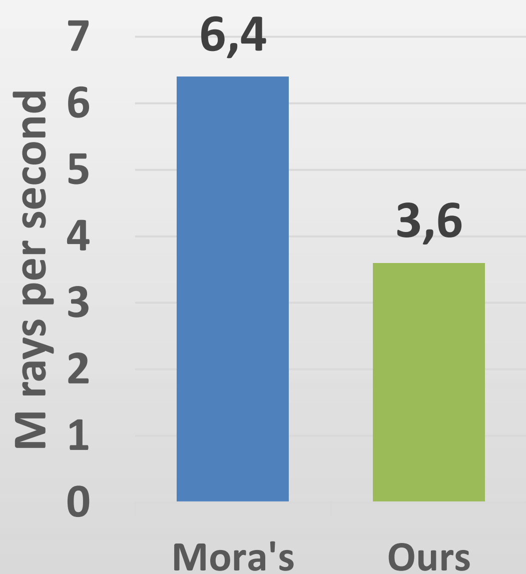
Performance Comparison to Mora's Method

- **Coherent rays using conic packets optimization**
  - **conic packets cannot be applied to secondary/random rays**
- **Incoherent rays for path tracing**
  - **our method outperforms Mora's method**

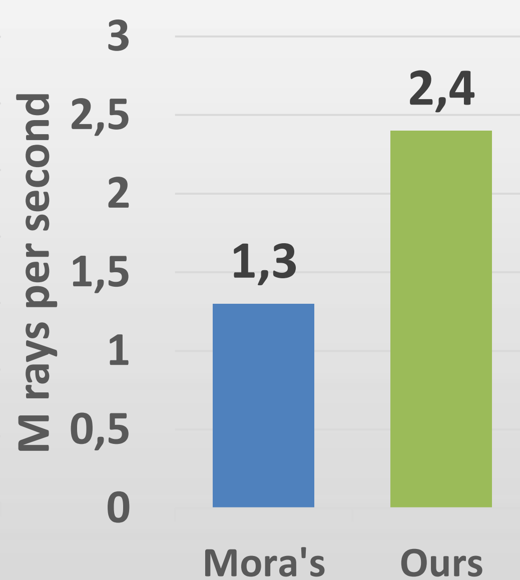
1.85 times faster!

point light path tracing

Mora's method	Ours
Core i7 3GHz	Core i7 2.67GHz



primary + shadow rays
one point light source



random rays
path tracing

Conclusions and Future Work

- Efficient DACRT algorithm using ray sampling
  - exploit distribution of rays for partitioning and traversal
  - derive cost metric to skip inefficient ray filtering
  - accelerate many types of rays by up to a factor of 2
    - reflection, ambient occlusion, area light, depth of field, path tracing
  - efficient for high resolution images with anti-aliasing
- Future work
  - multi-threading implementation
  - GPU implementation

Acknowledgements

- HPG reviewers for useful comments
- Funding
  - JSPS KAKEN Grant Number 24700093 & 13324688