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

Abstract

Divide-and-conquer ray tracing (DACRT) methods solve intersection problems between large numbers of rays and primitives by recursively subdividing the problem size until it can be easily solved. Previous DACRT methods subdivide the intersection problem based on the distribution of primitives only, and do not exploit the distribution of rays, which results in a decrease of the rendering performance especially for high resolution images with anti-aliasing. We propose an efficient DACRT method that exploits the distribution of rays by sampling the rays to construct an acceleration data structure. To accelerate ray traversals, we have derived a new cost metric which is used to avoid inefficient subdivision of the intersection problem where the number of rays is not sufficiently reduced. Our method accelerates the tracing of many types of rays (primary rays, less coherent secondary rays, random rays for path tracing) by a factor of up to 2 using ray sampling.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

Keywords: ray tracing, divide-and-conquer algorithm, ray sampling

1 Introduction

Recent advances in ray tracing have led to the achievement of interactive performance for both static and dynamic scenes, by using acceleration data structures such as grids [Wald et al. 2006; Kalojanov et al. 2011], kd-trees [Shevtsov et al. 2007; Zhou et al. 2008], and bounding volume hierarchies (BVHs) [Wald 2007]. These methods construct and store acceleration data structures before ray tracing.

In recent years, alternative approaches that do not store auxiliary data structures have been proposed [Mora 2011; Keller and Wachter 2011; Afra 2012]. These approaches, called divide-and-conquer ray tracing (DACRT), calculate the intersection tests between a set of rays and a set of primitives based on the divide-and-conquer algorithm. DACRT solves an intersection problem between a set of rays and a set of primitives by subdividing the problem into subproblems whose problem sizes (i.e. the numbers of rays and primitives) are smaller. DACRT subdivides the intersection problem by partitioning the set of primitives, and simultaneously traversing the set of rays.

Previous DACRT methods subdivide the problem based on the distribution of primitives only, which may not reduce the number of rays in the subproblem. This results in a waste of computational time on ray traversal, which is inefficient especially for a large number of rays. Recent demands on high resolution images require a large number of rays for ray tracing. Moreover, anti-aliasing methods that are used to improve the image quality usually require multi sampling and super sampling, resulting in the generation of a large number of rays.

To address this problem, our method exploits the distribution of rays by tracing a small subset of rays, that is, a sample of the rays. The main contribution of our method is that it derives a new cost metric that can be used to detect and avoid the inefficient subdivision of problems where the number of rays has not been sufficiently reduced. Our cost metric can be represented by a simple formula and easily estimated. Before partitioning the set of primitives and constructing a hierarchical acceleration data structure for the primitives, a small subset of rays is traced and information about the rays obtained (e.g. the ratios of rays intersecting each bounding volume of the partitioning candidates of the set of primitives).

To partition the primitives efficiently, the distribution of rays as well as the distribution of primitives is utilized, whereas the previous methods calculate the partitioning by assuming a uniform distribution of rays. In addition, the distribution of rays is utilized to determine the traversal order. Our experimental results demonstrate that the tracing of many types of rays such as primary rays, less coherent secondary rays (e.g. rays for specular reflection, ambient occlusion, area light source, depth of field), and random rays, can be accelerated up to a factor of 2 using our ray sampling method.

The contributions of our method are as follows.

- A concept of ray sampling is introduced. Our ray sampling method during acceleration structure construction and traversal improves the performance of tracing coherent rays, less coherent secondary rays, and incoherent rays.
- A new cost estimator is derived to avoid the inefficient subdivision of intersection problems in which the number of rays is not sufficiently reduced at the cost of intersection tests. The cost estimator is simple and therefore our method can easily be incorporated into the original DACRT method.

The performance gain of our method increases as the number of rays increases, which is beneficial for high resolution images and anti-aliasing using super sampling and multi sampling.

2 Previous Work

Many previous ray tracing methods construct and store acceleration data structures before tracing of the rays. Uniform grids [Fujimoto et al. 1986], kd-trees [Bentley 1975], and BVHs [Rubin and Whitted 1980] are widely used for acceleration data structures. For static scenes, the acceleration data structure can be precomputed and the computational time for this does not impact on the rendering frame rate. On the other hand, for dynamic scenes, the acceleration data structure must be built from scratch or updated for each frame. Therefore, the construction time and the time for ray tracing impact on the rendering frame rate, and fast construction methods

*e-mail:s141044@center.wakayama-u.ac.jp
†e-mail:iwasaki@sys.wakayama-u.ac.jp

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
HPG 2013, July 19 – 21, 2013, Anaheim, California.
Copyright © ACM 978-1-4503-2135-8/13/07 $15.00

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

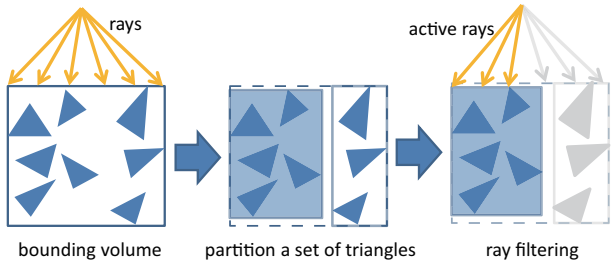


Figure 1: *An overview of DACRT. A single partitioning step (center) and ray filtering (right).*

for grids [Wald et al. 2006; Lagae and Dutré 2008; Kalojanov and Slusallek 2009], kd-trees [Hunt et al. 2006; Shevtsov et al. 2007; Zhou et al. 2008; Wu et al. 2011], and BVHs [Wald et al. 2007; Lauterbach et al. 2009; Garanzha et al. 2011] have been proposed.

In recent years, a number of divide-and-conquer ray tracing (DACRT) methods have been proposed [Keller and Wachter 2011; Mora 2011]. DACRT has several advantages compared to previous ray tracing methods. Firstly, DACRT does not need to store acceleration data structures, and the memory footprint can be determined in advance as a linear function of the number of primitives and the number of rays. This is beneficial for rendering hardware with limited memory. Secondly, the rendering speed of DACRT [Mora 2011] is faster than previous ray tracing methods for dynamic scenes, and is competitive for static scenes.

Mora's method has been highly optimized for primary rays by using conic packets. With conic packets, however, it is difficult to handle rays from different origins such as reflected or refracted rays, rays from area light sources, and random rays, while our method can accelerate tracing of those less coherent rays. Afra [Afra 2012] extended Mora's method to handle incoherent rays efficiently by using SSE (Streaming SIMD Extensions) and AVX (Advanced Vector Extensions) instructions. This method outperforms Mora's method for incoherent rays. Unfortunately, neither of these methods exploits the distribution of the rays. Our method further improves Afra's method by up to a factor of 2 for primary and less coherent rays, and consistently outperforms that method for incoherent rays.

Several methods that exploit the distribution and intersection results of a representative set of rays in order to construct acceleration data structures and ray traversals have been proposed. Bittner and Havran proposed a method to construct acceleration data structures using the distribution of the rays [Bittner and Havran 2009]. They derived the ray distribution heuristics (RDH) to determine the partitioning positions of hierarchical data structures. This method, however, increases the time required to construct the acceleration data structures and does not provide measurable performance gains in the time required for ray tracing. Feltman et al. [Feltman et al. 2012] derived the shadow ray distribution heuristic to accelerate the tracing of shadow rays by using representative shadow ray sets. Although this method can reduce the number of shadow ray traversals, the time to construct the acceleration data structures increases and additional memory is required, whereas our method does not increase the time required to construct the acceleration data structures compared to that without ray sampling. Moreover, the ray sampling overhead is negligible and this can be overcome by performance gains obtained through ray sampling.

3 Divide-And-Conquer Ray Tracing

DACRT performs the intersection tests between a set of rays and a set of primitives in a scene based on the divide and conquer algorithm, as illustrated in Fig. 1. The divide and conquer algorithm recursively breaks down the problem to be solved until it can be solved easily. For DACRT, the problem to be solved is the intersection tests between a set of primitives and a set of rays. Although DACRT can deal with arbitrary primitives, triangles are usually used as primitives. DACRT recursively partitions the set of triangles into subsets and partitions the set of rays that intersect the bounding volume (BV) of the set of triangles, until the number of triangles or the number of rays is sufficiently small for the intersection tests between the triangles and the rays to be easily calculated. DACRT partitions the set of triangles by using a tree structure such as kd-trees [Mora 2011] or BVHs [Afra 2012]. Leaf nodes of the tree structure store the triangles and each interior node of the tree structure is associated with a bounding volume that encloses the triangles in the descendant leaf nodes. Partitioning of the set of triangles affects the ray tracing performance. To determine an efficient partitioning, the cost function of the node associated with the bounding volume V is defined as follows.

$$C(V \rightarrow \{V_L, V_R\}) = C_T + C_I(p_L N_L + p_R N_R), \qquad (1)$$

where V_L and V_R are the bounding volumes of the child nodes, C_T is the cost of an intersection test between a ray and a node, C_I is the cost of an intersection test between a ray and a triangle, and p_L and p_R are the probabilities of rays intersecting V_L and V_R, respectively. N_L and N_R are the numbers of triangles included in V_L and V_R, respectively. Efficient partitioning is usually obtained by minimizing the cost function with Surface Area Heuristics (SAH). Since p_L and p_R are unknown in the construction, SAH assumes that the rays are uniformly distributed and approximates $p_L(p_R)$ by the ratio of the surface area of $V_L(V_R)$ to that of V.

Then DACRT calculates the set of rays that intersect each subdivided bounding volume. This operation is called *ray filtering*. Ray filtering requires an intersection test between the bounding volume and the rays. The filtered rays that intersect the bounding volume are called *active rays* as shown in Fig. 1.

4 Our Method

Our method uses ray sampling to improve DACRT. For the hierarchical data structure, a BVH represented by a binary tree is employed, while our method can also be applied to kd-trees and BVHs represented by n-ary trees [Dammertz et al. 2008]. Our method employs axis aligned bounding boxes (AABBs) for the bounding volume.

Algorithm 1 shows the pseudo code of our algorithm. The algorithm takes a bounding volume V, a set of active rays R, and a set of triangles T as inputs, and calculates intersection points between R and T. Our method first determines whether the current node is a leaf node or not, by checking the numbers of active rays and triangles. If the current node is not a leaf node, our method subdivides V into two bounding volumes by using the binning-based method [Wald 2007]. The binning-based method subdivides the bounding volume into K bins (equally-sized AABBs) $\{B_1, B_2, \cdots, B_K\}$. The bin that includes the center of each triangle's AABB is calculated. The set of triangles T in the bounding volume V can be divided into two disjoint subsets; $T_{L,j}$ in $\{B_1 \cdots B_j\}$ and $T_{R,j}$ in $\{B_{j+1} \cdots B_K\}$ for the j-th bin $B_j (1 \leq j \leq K-1)$. Let $V_{L,j}$ and $V_{R,j}$ be the bounding volumes of $T_{L,j}$ and $T_{R,j}$, respectively.

We now describe the overview of main steps in Algorithm 1. Our algorithm consists of four steps, ray sampling, partitioning, determining the traversal order, and traversal of child nodes with skipping ray filtering. In the ray sampling step, our method traces a small subset of active rays and calculates intersection tests between the subset of active rays and $V_{L,j}$ and $V_{R,j}$. In the intersection test,

Algorithm 1 Our Divide-And-Conquer Ray Tracing Algorithm. V is a bounding volume, R is a set of active rays, and T is a set of triangles in V. NaiveRT calculates intersection points between R and T. v_L and v_R are arrays for $V_{L,j}$, $V_{R,j}$, respectively. N_r, N_t, and N_s are the numbers of active rays R, triangles T, and sample rays, respectively.

```
procedure DACRT(V, R, T)
    if N_r is small or N_t is small then
        return NAIVERT(R, T)
    end if
    subdivide V into K bins {B_1, B_2, ··· , B_K}
    calculate T_{L,j}, T_{R,j}, V_{L,j}, and V_{R,j} for each bin B_j
    initialize arrays c_L, c_R, n_L, n_R ← {0}
    for each sample ray r do                          ▷ ray sampling
        INTERSECT(r, v_L, v_R, c_L, c_R, n_L, n_R)
    end for
    C_min ← ∞, j_min ← 1
    for j = 1 to K − 1 do    ▷ partitioning using cost function
        α_{L,j} ← c_{L,j}/N_s, α_{R,j} ← c_{R,j}/N_s
        calculate cost function C in Eq. (2)
        if C ≤ C_min then
            j_min ← j, C_min ← C
        end if
    end for
    if n_{L,j_min} ≥ n_{R,j_min} then        ▷ determine traversal order
        (V_0, α_0, T_0) ← (V_{L,j_min}, α_{L,j_min}, T_{L,j_min})
        (V_1, α_1, T_1) ← (V_{R,j_min}, α_{R,j_min}, T_{R,j_min})
    else
        (V_0, α_0, T_0) ← (V_{R,j_min}, α_{R,j_min}, T_{R,j_min})
        (V_1, α_1, T_1) ← (V_{L,j_min}, α_{L,j_min}, T_{L,j_min})
    end if
    if α_0 > 1 − C_bv/(n_child C_child) then
        DACRT(V_0, R, T_0)                            ▷ skip ray filtering
    else
        R_0 ← R ∩ V_0, DACRT(V_0, R_0, T_0)           ▷ ray filtering
    end if
    if α_1 > 1 − C_bv/(n_child C_child) then
        DACRT(V_1, R, T_1)                            ▷ skip ray filtering
    else
        R_1 ← R ∩ V_1, DACRT(V_1, R_1, T_1)           ▷ ray filtering
    end if
end procedure
```

our method stores informations about rays and bounding volumes, which are used in the latter steps. In the second step, our method estimates the cost function for $K-1$ partitioning candidates and the partitioning that provides the minimum cost is selected. In the third step, our method determines the traversal order of two child nodes based on the informations obtained in the ray sampling step. In the fourth step, we first determine whether the ray filtering reduces the computational cost or not, by using a simple criterion. Then our method recursively traverses the child nodes.

4.1 Ray sampling

Active rays are sampled to calculate the cost function, the distribution of active rays in the bounding volume, and the traversal order of child nodes. The sampled rays are referred to as *sample rays*. In the current implementation, the number of sample rays is fixed (100) and is irrespective of the number of active rays. The sample rays are selected randomly when the number of active rays is more than 1000. Otherwise our method avoids ray sampling since the overhead of ray sampling cannot be amortized.

Algorithm 2 Intersection tests between sample ray r and $V_{L,j}$ and $V_{R,j}$. INTERSECTP(V, r, t_n, t_f) returns true if ray r intersects V, and sets the intersection interval between r and V to $[t_n, t_f]$. INTERSECTP is a common ray-box intersection algorithm as shown in [Pharr and Humphreys 2010].

```
procedure INTERSECT(r, v_L, v_R, c_L, c_R, n_L, n_R)
    for j = 1 to K − 1 do
        d_{L,j}, d_{R,j} ← ∞
        if INTERSECTP(V_{L,j}, r, t_n, t_f) then
            c_{L,j} ← c_{L,j} + 1, d_{L,j} ← t_n
        end if
        if INTERSECTP(V_{R,j}, r, t_n, t_f) then
            c_{R,j} ← c_{R,j} + 1, d_{R,j} ← t_n
        end if
        if d_{L,j} < d_{R,j} then
            n_{L,j} ← n_{L,j} + 1
        else
            n_{R,j} ← n_{R,j} + 1
        end if
    end for
end procedure
```

In the ray sampling step, our method calculates intersection tests between each sample ray and all pairs of $V_{L,j}$ and $V_{R,j}$ as shown in Algorithm 2. In the intersection test, our method calculates numbers of sample rays, $c_{L,j}$ and $c_{R,j}$, that intersect $V_{L,j}$ and $V_{R,j}$, respectively. In addition, for each pair $V_{L,j}$ and $V_{R,j}$, the bounding volume closer to each sample ray is determined in Algorithm 2. To determine the closer bounding volume, our method compares the entry distances (t_n in Algorithm 2) to $V_{L,j}$ and $V_{R,j}$ for each sample ray. Our method calculates the numbers of sample rays, $n_{L,j}$ and $n_{R,j}$ in Algorithm 2, closer to $V_{L,j}$ and $V_{R,j}$, respectively.

4.2 Partitioning using Cost function

The SAH cost function approximates the probability of a node being hit by a ray as the ratio of the surface area of the child node to that of the parent node. This approximation works well when the rays are uniformly distributed in the scene, while it may not provide a good estimation when the ray distribution is concentrated. Since previous methods construct acceleration data structures before ray tracing, the probability of a node being hit by a ray cannot be estimated.

On the other hand, since DACRT constructs a BVH and traces rays simultaneously, the probability can be estimated by using the numbers of sample rays, $c_{L,j}$ and $c_{R,j}$, that intersect $V_{L,j}$ and $V_{R,j}$. Our method calculates the ratio, $\alpha_{L,j}(\alpha_{R,j})$, of sample rays intersecting $V_{L,j}(V_{R,j})$. We refer to this ratio as the *intersection ratio*. The probabilities p_L and p_R in Eq (1) are estimated by using the intersection ratios $\alpha_{L,j}$ and $\alpha_{R,j}$. Then the cost function C is calculated by:

$$C(V \rightarrow \{V_{L,j}, V_{R,j}\}) = C_T + C_I(\alpha_{L,j} N_{L,j} + \alpha_{R,j} N_{R,j}), \quad (2)$$

where $N_{L,j}$ and $N_{R,j}$ are the numbers of triangles in $V_{L,j}$ and $V_{R,j}$, respectively. By using this cost function, our method can consider the actual distribution of rays to construct the BVH.

4.3 Traversal order

Previous method [Afra 2012] determines the traversal order of two child nodes by using a single active ray. While this estimation works well for incoherent rays, we observed that this estimation is inefficient for tracing secondary rays and random rays with a small

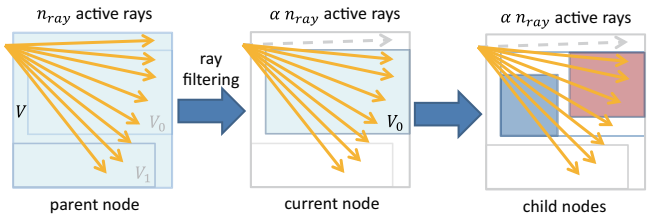


Figure 2: *Inefficient case of ray filtering. The number of active rays (orange color) reduced by ray filtering is only one in this case. The cost for intersection tests between active rays of parent node and V_0 is $n_{ray}C_{bv}$. The cost for intersection tests between active rays intersecting V_0 and bounding volumes of n_{child} child nodes is estimated as $\alpha n_{ray}C_{child}n_{child}$, where α is the intersection ratio of V_0.*

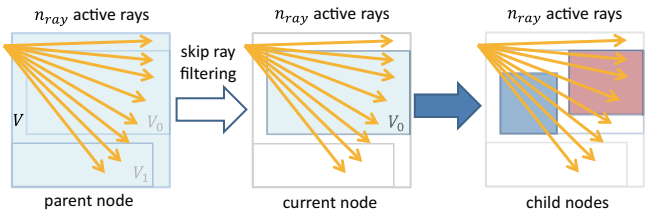


Figure 3: *Skip ray filtering if the number of active rays is not sufficiently reduced. The computational cost C_{skip} for skipping ray filtering is that of the intersection tests between n_{ray} active rays of current node and child nodes, and estimated as $n_{ray}C_{child}n_{child}$.*

number of bounces, and the inefficient order traversal decreases the ray tracing performance. To address this problem, our method uses the number of sample rays closer to the bounding volume of each child node, which is obtained in the ray sampling step. Since the calculation of entry distances is necessary to obtain the intersection ratios, the additional operation to determine the efficient traversal order is only a comparison, whose computational cost is negligible compared to the performance loss due to inefficient order traversals.

4.4 Skip ray filtering

The ray filtering calculates the intersection tests between active rays intersecting V and the bounding volumes V_0 and V_1 as listed in Algorithm. 1. Since this step calculates active rays for V_0 and V_1, we now consider the node with V_0 as current node, and that with V as parent node for the explanation. As shown in Fig. 2, when most of the active rays of parent node intersect the bounding volume of current node during ray filtering, the problem size (i.e. the number of rays) is not sufficiently divided and this wastes a huge amount of computational time on intersection tests between the bounding volume of current node and the active rays of parent node. In this case, it is efficient to skip the ray filtering and assume that all active rays intersect the bounding volume.

To determine whether or not to skip ray filtering, we propose a new cost metric that uses the intersection ratio α of the current node, which is obtained in the ray sampling step. The cost metric is the sum of two costs, the cost of intersection tests between the bounding volume of the current node and the active rays of parent node, and the cost of intersection tests between the child nodes and the active rays of the current node. Let n_{ray} be the number of active rays of the parent node, C_{bv} the cost of an intersection test between a bounding volume and a single ray. The cost of the intersection tests between n_{ray} rays and the bounding volume of current node is given by $n_{ray}C_{bv}$. The number of active rays for the current node is estimated as αn_{ray}. Our method then calculates the cost of intersection tests between the child nodes and the active rays of the current node. Let C_{child} be the cost of the intersection test between a child node and a single ray, and n_{child} be the number of child nodes. The cost of intersection tests between n_{child} child nodes and αn_{ray} rays is given by $\alpha n_{ray}C_{child}n_{child}$. Thus, the total cost C_{int} with ray filtering can be calculated from the following equation.

$$C_{int} = n_{ray}C_{bv} + \alpha n_{ray}C_{child}n_{child}. \tag{3}$$

Next, we describe the cost C_{skip} of skipping the intersection tests as illustrated in Fig. 3. In this case, since our method skips the intersection tests, the cost of the intersection tests between the active rays of parent node and the bounding volume of current node is zero and n_{ray} active rays remain. The cost of intersection tests between n_{child} child nodes and n_{ray} active rays is $n_{ray}C_{child}n_{child}$. Therefore the total cost C_{skip} without ray filtering is calculated by:

$$C_{skip} = n_{ray}C_{child}n_{child}. \tag{4}$$

Our method skips the ray filtering if $C_{int} > C_{skip}$. That is, the skipping criterion for the intersection ratio α is calculated by the following equation.

$$\alpha > 1 - \frac{C_{bv}}{n_{child}C_{child}}. \tag{5}$$

If the current node is a leaf node, the cost C_{child} is the cost of an intersection test between a triangle and a ray, and n_{child} is the number of triangles included in the leaf node. If the current node is not a leaf node, the cost C_{child} is equal to the cost C_{bv}, and n_{child} is 2 since in our method the BVH is represented by a binary tree. In summary, the skipping criterion for a non-leaf node is simplified to the following equation.

$$\alpha > 0.5. \tag{6}$$

5 Results

We tested several scenes on a standard PC with an Intel Core i7 2.67GHz CPU and 6GB memory. The timings listed in Figs. 4, 5, 6, and 7, are those for ray tracing and do not include ray generation and shading, and measured in a single thread. SSE is utilized for efficient ray tracing. In order to exploit SSE, the data structures of the rays and the triangles are similar to Afra's method [Afra 2012]. To demonstrate the effectiveness of our method, we compare the computational time for ray tracing with and without ray sampling [Afra 2012]. In the latter case, the partitioning is determined by using the SAH cost function. To perform a fair comparison, our method adaptively uses our cost function and middle partitioning similar to Afra's method. If the ratio of the number of active rays and that of triangles is larger than a threshold (1.5 in our method), the set of triangles is partitioned using the cost function described in Sec. 4.2. Our method uses 32 bins and obtains good results.

Fig. 4 shows the Sibenik scene rendered with different viewpoints and different image resolutions. The floor is a specular surface with one point light source (Figs. 4(a)(b)) or an area light source (Fig. 4(c)). As shown in Fig. 4, our method gives an average increase in speed of 1.85 and accelerates ray tracing by up to a factor 2 for primary, secondary, and shadow rays from point and area light sources. To handle secondary rays, our method first traces primary rays from the viewpoint and calculates the hit points. The reflected rays are generated and traced from the hit points on the specular

surfaces. For secondary rays, our method rebuilds the BVH from scratch and does not reuse the BVH constructed for the primary rays. We have experimented with the reuse of the BVH for the secondary rays for the Sibenik model. However, the reuse of the BVH does not provide meaningful performance gains compared to our method that reconstructs the BVH for the secondary rays, since the computational time for tracing of rays is dominant compared to that for constructing the BVH for the Sibenik model.

We measured the impacts of, first, using the cost function given in Eq. (2), secondly, traversal order determination using sample rays, and thirdly, skipping the ray filtering for the 4096^2 resolution image in Fig. 4(a). The acceleration ratios obtained by use of the cost function, traversal order, and skipping ray filtering were 1.24×(27.3s/22.0s,42%), 1.05×(27.3s/26.0s,10%), and 1.28×(27.3s/21.3s,48%), respectively, where the numbers in the parentheses are the computational times in seconds without and with each technique, and the percentage contribution made to the total reduced time. As shown in this measurement, the performance gains due to skipping ray filtering and the cost function using actual ray distributions have the most impact.

Fig. 5(a) shows a case for which our method is not so efficient. In this case, the performance gain due to ray sampling is relatively small for moderate resolution images and the computational time without ray sampling is slightly smaller than that with ray sampling for the 1024^2 image. However, the performance loss due to ray sampling is small (3ms) and our method becomes faster for high resolution images. Fig. 5(b) shows the results of rendering with ambient occlusion (AO). For AO rays, our method traces a random ray on the hemisphere of each hit point. Fig. 5(c) shows the San Miguel scene with depth of field (DOF) rendering with path tracing up to 3 bounces.

Figs. 6 and 7 show the Conference and Sponza scenes, respectively. Fig. 8 shows the acceleration ratios using ray sampling with different image sizes and different bounce numbers of rays. As shown in Fig. 8(a), the acceleration ratio increases as the number of rays increases. That is, our method is suited to high resolution images as well as moderate resolution images with multi-sampled anti-aliasing (MSAA). In fact, all the images are rendered in 512^2 resolution with MSAA (e.g. a 4096^2 resolution image is rendered as a 512^2 image with 64 MSAA). Fig. 8(b) shows that our method can accelerate the ray tracing for coherent and incoherent rays.

We compared the performance of tracing coherent rays with that of Mora's method [Mora 2011] for the Conference scene in Fig. 6(a). Our method is 1.66 fps for 1024×1024 image on 2.67GHz CPU, while Mora's cone optimization method is 3.3 fps for 1280×800 image on 3GHz CPU. For the primary rays and shadow rays of one point light source, our method is 2 times slower than Mora's method that is optimized for primary rays. However, as described in Sec. 2, Mora's cone optimization cannot be applied to secondary rays and random rays, while our method can accelerate tracing of those rays. Therefore the applicability of our method is much wider than that of Mora's method.

For the Conference scene in Fig. 6(c), we compared the performance in terms of millions of rays per second with those reported for previous DACRT methods [Mora 2011; Afra 2012]. The performance for path tracing of diffuse rays with 8 bounces (no Russian roulette) was 2.4 (M rays/s) using our method, 2.0 using Afra's method, and 1.3 using Mora's method. Although our method was measured on the slowest clocked Intel Core i7 CPU (ours : 2.67GHz, Afra : 3.4GHz, Mora : 3GHz), it outperforms the previous DACRT methods for incoherent rays (1.20×Afra's method and 1.85×Mora's method). For incoherent rays, the distribution of rays tends to be uniform and SAH provides a good estimation. However our method is still faster compared to Afra's method, which is optimized for incoherent rays. Although our method is slower than Mora's method for primary rays, our method is at least 1.85 times faster for incoherent rays. Therefore, we think our method outperforms Mora's method in the total computational times in path tracing where the computation of incoherent rays is dominant.

6 Conclusions and Future Work

We have presented an efficient DACRT algorithm using ray sampling. Our method exploits the distribution of rays to construct a BVH and determine an efficient traversal order. We have derived a new cost metric that can skip inefficient subdivisions by using the intersection ratio. The derived metric for non-leaf nodes of binary BVHs is simplified and easily calculated using the intersection ratio. The ray sampling used in our method can accelerate the tracing of many types of rays by up to a factor of 2. The method is most efficient for high resolution images with high quality anti-aliasing.

In future work, we would like to implement our method using multithreading. We also intend to apply our method to the motion blur.

Acknowledgements

We would like to thank all the reviewers for their useful comments. This research was partially supported by JSPS KAKENHI Grant Number 24700093 and 13324688.

References

AFRA, A. T. 2012. Incoherent ray tracing without acceleration structures. In *Proc. of Eurographics Short Paper*, 97–100.

BENTLEY, J. L. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9, 509–517.

BITTNER, J., AND HAVRAN, V. 2009. RDH: ray distribution heuristics for construction of spatial data structures. In *Proc. of the 25th Spring Conference on Computer Graphics*, 51–58.

DAMMERTZ, H., HANIKA, J., AND KELLER, A. 2008. Shallow bounding volume hierarchies for fast simd ray tracing of incoherent rays. *Computer Graphics Forum 27*, 4, 1225–1233.

FELTMAN, N., LEE, M., AND FATAHALIAN, K. 2012. SRDH: Specializing BVH construction and travesal order using representative shadow ray sets. In *Proc. of High Performance Graphics 2012*, 49–55.

FUJIMOTO, A., TANAKA, T., AND IWATA, K. 1986. Arts: Accelerated ray tracing system. *IEEE Computer Graphics and Applications 6*, 4, 16–26.

GARANZHA, K., PANTALEONI, J., AND MCALLISTER, D. 2011. Simpler and faster HLBVH with work queues. In *Proc. of High Performance Graphics 2011*, 59–64.

HUNT, W., MARK, W. R., AND STOLL, G. 2006. Fast kd-tree construction with an adaptive error-bounded heuristic. In *Proc. of IEEE Symposium on Interactive Ray Tracing*, 81–88.

KALOJANOV, J., AND SLUSALLEK, P. 2009. A parallel algorithm for construction of uniform grids. In *Proc. of High Performance Graphics 2009*, 23–28.

KALOJANOV, J., BILLETER, M., AND SLUSALLEK, P. 2011. Two-level grids for ray tracing on GPUs. *Computer Graphics Forum 30*, 2, 307–314.

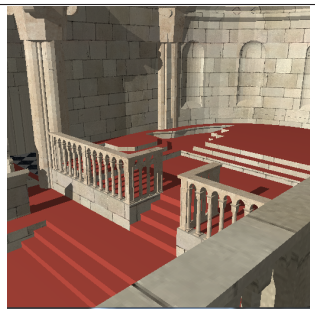
Sibenik			
image size	(a) point light	(b) point light	(c) area light
512^2	415ms/295ms (1.41)	281ms/183ms (1.54)	374ms/255ms (1.47)
1024^2	1520ms/1003ms (1.52)	1130ms/700ms (1.61)	1430ms/860ms (1.66)
2048^2	5992ms/3759ms (1.59)	4656ms/2647ms (1.76)	5730ms/3164ms(1.81)
4096^2	27300ms/14700ms (1.86)	19772ms/11330ms (1.75)	22269ms/11491ms (1.94)

Figure 4: *Computational times in milliseconds of ray tracing without and with ray sampling (our method) for different image sizes of the Sibenik scene (75K triangles). The number in the parentheses is the acceleration ratio using our method.*

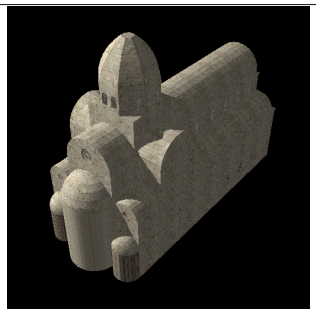
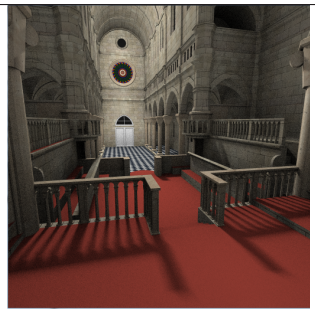
			
image size	(a) point light	(b) AO	(c) PT and DOF
512^2	110ms/110ms (1.00)	446ms/365ms (1.22)	17319ms/16876ms (1.03)
1024^2	331ms/334ms (0.99)	1716ms/1217ms (1.41)	27725ms/26435ms (1.05)
2048^2	1236ms/1130ms (1.09)	6948ms/4359ms (1.59)	67253ms/59893ms (1.12)
4096^2	5075ms/4219ms (1.20)	29439ms/19300ms (1.53)	215805ms/173018ms (1.25)

Figure 5: *Computational times in milliseconds for (a) inefficient cases, (b) rendering with ambient occlusion (AO), and (c) depth of field (DOF) rendering with path tracing (PT) for the San Miguel scene (7.9M triangles). The acceleration ratio using our method is listed in the parenthesis.*

KELLER, A., AND WACHTER, C. 2011. Efficient ray tracing without auxiliary acceleration data structure. In *Proc. of High Performance Graphics 2011 (Poster)*.

LAGAE, A., AND DUTRÉ, P. 2008. Compact, fast and robust grids for ray tracing. *Computer Graphics Forum 27*, 4, 1235–1244.

LAUTERBACH, C., GARLAND, M., SENGUPTA, S., LUEBKE, D., AND MANOCHA, D. 2009. Fast BVH construction on GPUs. *Computer Graphics Forum 28*, 2, 375–384.

MORA, B. 2011. Naive ray-tracing: A divide-and-conquer approach. *ACM Trans. Graph. 30*, 5, 117:1–117:12.

PHARR, M., AND HUMPHREYS, G. 2010. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers.

RUBIN, S. M., AND WHITTED, T. 1980. A 3-dimensional representation for fast rendering of complex scenes. *SIGGRAPH Comput. Graph. 14*, 3, 110–116.

SHEVTSOV, M., SOUPIKOV, A., AND KAPUSTIN, E. 2007. Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *Computer Graphics Forum 26*, 3, 395–404.

WALD, I., IZE, T., KENSLER, A., KNOLL, A., AND PARKER, S. G. 2006. Ray tracing animated scenes using coherent grid traversal. *ACM Trans. Graph. 25*, 3, 485–493.

WALD, I., BOULOS, S., AND SHIRLEY, P. 2007. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph. 26*, 1, 6:1–6:18.

WALD, I. 2007. On fast construction of SAH-based bounding volume hierarchies. In *Proc. of the 2007 IEEE Symposium on Interactive Ray Tracing*, 33–40.

WU, Z., ZHAO, F., AND LIU, X. 2011. SAH kd-tree construction on GPU. In *Proc. of High Performance Graphics*, 71–78.

ZHOU, K., HOU, Q., WANG, R., AND GUO, B. 2008. Real-time kd-tree construction on graphics hardware. *ACM Trans. Graph. 27*, 5, 126:1–126:11.

Conference			
image size	(a) point light	(b) point light	(c) path tracing
512^2	234ms/189ms (1.24)	273ms/189ms (1.44)	1208ms/1071ms (1.13)
1024^2	803ms/602ms (1.34)	1030ms/671ms (1.54)	3936ms/3279ms (1.20)
2048^2	3113ms/2182ms (1.43)	5992ms/2636ms (1.60)	15533ms/12316ms (1.26)
4096^2	12784ms/8930ms (1.43)	17987ms/10618ms (1.69)	65117ms/47761ms (1.36)

Figure 6: *Computational times in milliseconds of ray tracing without and with ray sampling (our method) for different image sizes of the Conference scene (331K triangles). The acceleration ratio using our method is listed in the parenthesis.*

Sponza			
image size	(a) point light	(b) point light	(c) path tracing
512^2	661ms/478ms (1.38)	599ms/448ms (1.52)	2853ms/2331ms (1.22)
1024^2	2298ms/1428ms (1.61)	2213ms/1471ms (1.50)	8288ms/7066ms (1.17)
2048^2	8801ms/5056ms (1.74)	8774ms/5363ms (1.64)	31960ms/23779ms (1.34)
4096^2	35397ms/19657ms (1.80)	35643ms/20548ms (1.73)	136327ms/98228ms (1.39)

Figure 7: *Computational times in milliseconds of ray tracing without and with ray sampling (our method) for different image sizes of the Sponza scene (262K triangles). The acceleration ratio using our method is listed in the parenthesis.*

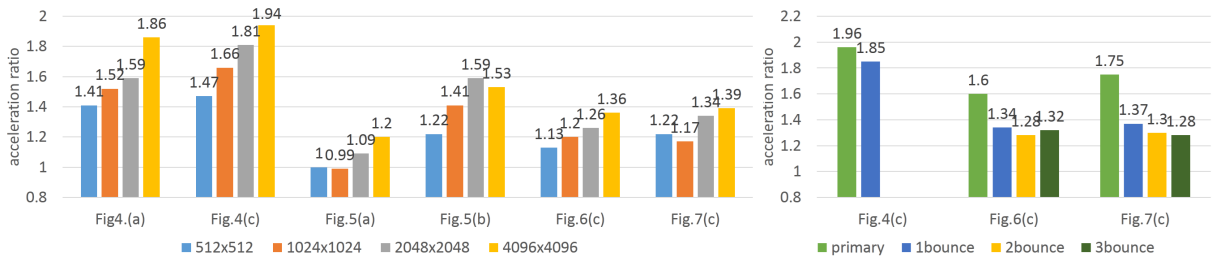


Figure 8: *Acceleration ratios using ray sampling for (a) different image sizes and (b) different bounce numbers of rays.*