

## PAPER

# A Fast Ray-Tracing Using Bounding Spheres and Frustum Rays for Dynamic Scene Rendering

Ken-ichi SUZUKI<sup>†a)</sup>, Member, Yoshiyuki KAERIYAMA<sup>††</sup>, Kazuhiko KOMATSU<sup>†††</sup>, Nonmembers,  
Ryusuke EGAWA<sup>†††</sup>, Nobuyuki OHBA<sup>††††</sup>, and Hiroaki KOBAYASHI<sup>†††</sup>, Members

**SUMMARY** Ray tracing is one of the most popular techniques for generating photo-realistic images. Extensive research and development work has made interactive static scene rendering realistic. This paper deals with interactive *dynamic* scene rendering in which not only the eye point but also the objects in the scene change their 3D locations every frame. In order to realize interactive dynamic scene rendering, RTRPS (Ray Tracing based on Ray Plane and Bounding Sphere), which utilizes the coherency in rays, objects, and grouped-rays, is introduced. RTRPS uses bounding spheres as the spatial data structure which utilizes the coherency in objects. By using bounding spheres, RTRPS can ignore the rotation of moving objects within a sphere, and shorten the update time between frames. RTRPS utilizes the coherency in rays by merging rays into a ray-plane, assuming that the secondary rays and shadow rays are shot through an aligned grid. Since a pair of ray-planes shares an original ray, the intersection for the ray can be completed using the coherency in the ray-planes. Because of the three kinds of coherency, RTRPS can significantly reduce the number of intersection tests for ray tracing. Further acceleration techniques for ray-plane-sphere and ray-triangle intersection are also presented. A parallel projection technique converts a 3D vector inner product operation into a 2D operation and reduces the number of floating point operations. Techniques based on frustum culling and binary-tree structured ray-planes optimize the order of intersection tests between ray-planes and a sphere, resulting in 50% to 90% reduction of intersection tests. Two ray-triangle intersection techniques are also introduced, which are effective when a large number of rays are packed into a ray-plane. Our performance evaluations indicate that RTRPS gives 13 to 392 times speed up in comparison with a ray tracing algorithm without organized rays and spheres. We found out that RTRPS also provides competitive performance even if only primary rays are used.

**key words:** computer graphics, ray tracing, intersection test, bounding volume, bounding sphere

## 1. Introduction

Ray tracing is an image rendering method based on the global illumination model, and it can generate photo-realistic images for many applications, such as product design, architecture, and entertainment. Ray tracing, however, requires a huge number of computations for intersection tests between rays and objects. In order to make the ray tracing faster, the spatial coherency of objects and rays has

been exploited.

The first spatial coherency is in objects in 3D space. Because objects in a scene do not exist with a constant density but concentrate in some small areas, they can be merged into a smaller number of groups. The number of intersection tests between objects and rays can be reduced by using the groups instead of the individual objects. The grouping of objects can be done by using space-subdivision techniques or bounding volumes. The former is known as kd-tree [1]–[6], uniform grid [7]–[9], and octree [10]. The latter is known as axis aligned bounding box [11] and oriented bounding box [12].

The second one, coherency in rays, has also been utilized, including beam-tracing [13], cone-tracing [14], and pencil-tracing [15]. Frustum ray-casting algorithms [16], [17] and a ray-packet technique [18] have also been presented. In these studies, rays are merged into a set of rays, exploiting the similarity in nearby rays. Because of the similarity, those rays in a set can share the spatial data structure and possible intersecting object data, resulting in a shorter calculation time.

Recently, using the coherency in objects and rays, Reshetov *et al.* [19] realized interactive *static* scene rendering. The term *static* means that the objects do not change their 3D locations in a series of frames, and only the eye point may move in the static object space.

However, the interactive static scene rendering has been available only for ray-tracing with primary rays (a.k.a. FHRT; First-Hit Ray Tracing). Then, the next challenge should be interactive *dynamic* scene ray tracing, where both the eye point and the objects move between frames, using *secondary rays* for photo-realistic effects. The secondary rays here include super-sampling rays at a diffuse surface and shadow rays from a light source\* as well as the reflective/refractive rays. Since most of the scenes in the world have plenty of diffuse surfaces and light sources, rendering of such a scene uses a huge number of secondary rays as shown in Fig. 1.

The difficulty in rendering a dynamic scene comes from the update cost of spatial data structure. In a dynamic scene, a lot of objects can change their locations, and therefore the spatial data structures cannot be reused but updated

Manuscript received April 23, 2009.

Manuscript revised November 14, 2009.

<sup>†</sup>The author is with Tohoku Institute of Technology, Sendai-shi, 982–8577 Japan.

<sup>††</sup>The author is with Nikon Corporation, Kumagaya-shi, 360–8559 Japan.

<sup>†††</sup>The authors are with Cyberscience Center, Tohoku University, Sendai-shi, 980–8578 Japan.

<sup>††††</sup>The author is with IBM Research, Tokyo Research Laboratory, IBM Japan Ltd., Yamato-shi, 242–8502 Japan.

a) E-mail: ksuzuki@tohotech.ac.jp

DOI: 10.1587/transinf.E93.D.891

\*Although the shadow rays are not actually 'secondary,' but we treat in this paper any rays other than primary rays directly from the eye point as secondary rays.

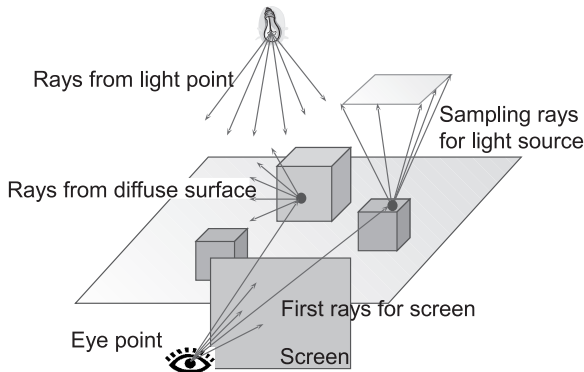


Fig. 1 Rays used in the ray-tracing process.

every frame. However, the conventional methods for static scene rely upon the reusability of the spatial data structure, and it takes a very long time to re-calculate the structure even if only a small number of objects move. Since the total rendering time is the sum of the update time of the spatial data structure and the rendering time per frame, a spatial data structure with low update cost is necessary.

Fast rendering using secondary rays is not easy, because the secondary rays have less spatial coherency than the primary ones. As shown in Fig. 1, from the eye point, the primary rays are shot to the screen. Therefore, the direction vectors of the nearby primary rays are quite similar and suitable for forming a group (e.g. ray-packet). On the other hand, secondary rays, especially the rays from a diffuse surface, are sometimes radiated approximately in the range of 180 degrees. For utilizing the coherency from the secondary diffuse rays, another ray grouping technique is required.

In this paper, we present a ray tracing scheme named RTRPS (Ray Tracing based on Ray-Plane and bounding Sphere) for dynamic scene rendering with a number of secondary rays. RTRPS uses bounding spheres for clustering the objects, considering the update cost between frames for dynamic scene rendering. As we discuss in Sect. 2.1, spheres can ignore the rotation of the objects in it, resulting in a shorter update time. In order to intensively exploit the spatial coherency of secondary diffuse rays, RTRPS assumes that the secondary rays are shot to a grid plane and builds a *ray-plane* from a series of rays. The produced ray-planes have a characteristic that a pair of orthogonal ray-planes can reproduce an original ray. From this characteristic, we can extract another coherency from the grouped rays, inter-ray-group coherency, which can reduce the number of intersection tests between rays and objects.

Note that, because it is difficult to build a ray-plane for the reflection/refraction rays due to their individual directions, RTRPS does not contribute to the speedup of the reflection/refraction processing. However, the number of reflection/refraction rays does not increase at the intersection points, resulting in the smaller total number of rays than that of diffuse rays.

This paper is organized as follows. Section 2 describes the concept of RTRPS using a spatial data structure con-

sisting of bounding spheres and ray-planes for ray clustering. Section 3 presents further acceleration techniques for RTRPS. Section 4 shows the performance evaluations. In Sect. 5, we conclude the paper with a summary of contributions.

## 2. RTRPS: Ray Tracing Based on Ray-Plane and Bounding Sphere

RTRPS has two distinctive features to achieve fast dynamic scene rendering with a number of secondary rays. The first is using *bounding spheres* as the spatial data structure and the second is building *ray-planes* from the radiated rays.

### 2.1 Bounding Spheres for Spatial Data Structure

Pre-computed spatial data structures are very useful for fast image rendering in static scenes, because they can eliminate a number of intersection tests between rays and objects based on the spatial coherency of objects, and can be reused for every frame. However, in the dynamic scene rendering, objects in a scene can move every frame, and therefore the pre-computed structure cannot be reused but must be updated. For this reason, in dynamic scenes, the update time between frames has a significant effect on the total time, as well as the rendering time in a frame.

The spatial data structures are categorized into two classes, space-subdivision and bounding volume. The former includes kd-tree, uniform grid and octree. For the latter, bounding boxes are commonly used. Havran *et al.* [20] evaluated the update time for various spatial data structures under the same condition, and showed that, for a shorter update time, the bounding volume techniques are better than the space-subdivision techniques. Therefore, we choose the bounding volume as a spatial data structure of dynamic scene rendering, rather than the space-subdivision techniques.

Bounding box and bounding sphere are well-known as simple bounding volumes. Although the bounding boxes are commonly used [11], [12], [21]–[28], RTRPS uses bounding *spheres* for the bounding volume. There are two reasons for this choice. The first reason is that the bounding sphere can ignore the rotation of the objects inside, resulting in a shorter update time. The second is that the intersection test between a bounding sphere and *ray-plane*, which will be discussed in the next subsection, can be simple. The rest of this subsection discusses the first reason, update time of bounding boxes and spheres.

The movement of an object can be classified based on the degree of freedom (DOF) as follows:

- (i) Translation (3 DOF)
- (ii) Rotation (3 DOF)
- (iii) Homothetic Transformation (1 DOF)
- (iv) Affine Transformation (5 DOF)
- (v) Projective Transformation (3 DOF).

If the movement of objects consists only of translation (i: 3 DOF), a bounding box containing the objects can be up-

dated simply by the relocation of its position vector. However, when the movement of objects includes both translation and rotation (i and ii: 6DOF), the width and height of the bounding box must be updated in addition to the translation of the bounding box. On the other hand, by using the bounding spheres, even in the case of (i and ii), only the translation of the sphere is necessary.

In this paper, we focus on the dynamic scenes containing above (i) and (ii) of object movement. Although these scenes do not handle the expansion and shrink of an object, most of typical dynamic scenes can be covered. The data structure using bounding spheres can be updated by a single affine transformation of the center of spheres, resulting in much shorter update time than those using bounding boxes and space-subdivision scheme.

## 2.2 Grouping the Rays into a Plane

Faster ray tracing has been achieved by exploiting the coherency in objects and rays. RTRPS utilizes the coherency in objects by using bounding spheres as a spatial data structure mentioned in the previous subsection. The coherency in rays is also utilized by building a *ray-plane* from the primary and secondary rays.

Many researchers have used a group of rays instead of handling individual rays [13]–[18] for utilizing the coherency in rays. However, simply using the groups of rays is not enough to reach our goal, interactive dynamic scene rendering with many secondary rays. In order to achieve a breakthrough, RTRPS assumes that the rays are radiated to a grid plane facing to the origin of the ray. Figure 2 shows an example of this assumption. Since the rays are radiated to the grid point, a row of rays can be merged into a *ray-plane*. The ray-planes form a row-plane group and a column-plane group orthogonal each other. The characteristic of ray-planes is that each set of a row-plane and column-plane shares an original ray, which has the coherency between ray-groups, i.e. inter-ray-group coherency. This kind of coherency has never been exploited in the conventional ray grouping techniques. Utilizing the inter-ray-group coherency, for  $N \times N$  rays,  $2N$  intersection tests are enough as shown in Fig. 2. Here, note that the inter-ray-group coherency can be utilized in addition to any acceleration methods available for frustums or cones, such as frustum culling. The acceleration techniques for the intersection test of ray-planes are presented in Sect. 3.

Because RTRPS uses bounding spheres for clustering objects, the intersection test procedure between a plane and sphere is essential. As shown in Fig. 3, let the center of a sphere be  $\mathbf{c}$ , the radius of the sphere  $r$ , a representative point on the plane  $\mathbf{p}_0$ , and the normal vector of the plane  $\mathbf{n}$ . Then the arbitrary point  $\mathbf{p}$  on the plane is given by the following equation:

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0 \quad \Leftrightarrow \quad \mathbf{n} \cdot \mathbf{p} + p_w = 0$$

where  $p_w = -\mathbf{n} \cdot \mathbf{p}_0$ . The distance between the plane and the center of the sphere  $\mathbf{c}$  can be calculated by

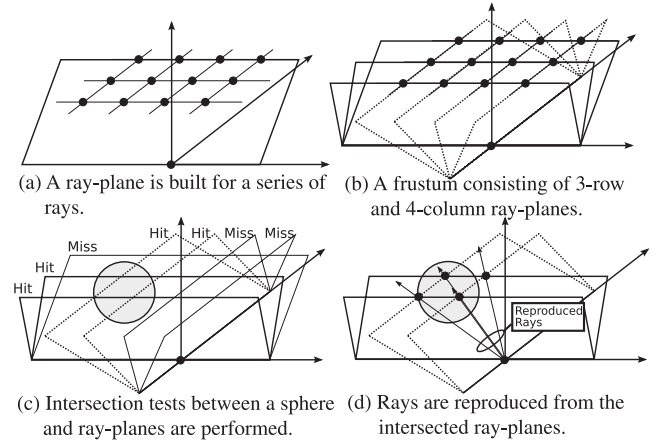


Fig. 2 Intersection test procedure of ray-planes and a bounding sphere.

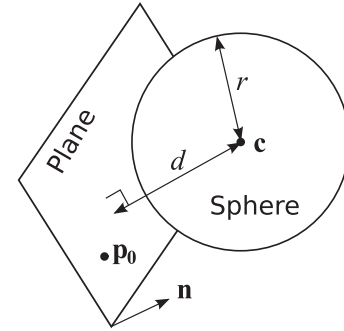


Fig. 3 Intersection test between a plane and sphere.

$$d = |\mathbf{n} \cdot \mathbf{c} + p_w| \quad (1)$$

If  $d < r$ , the plane and bounding sphere intersect with each other. Here, note that  $p_w$  has a fixed value when the ray-plane is built, and it can be reused for all the spheres. It means that a *plane*-bounding-sphere intersection test can be completed by only one vector inner product calculation and a subtraction (eventually it can be just a single 2D inner product as shown in Sect. 3.1.1). On the other hand, a *plane*-bounding-*box* intersection test requires many conditional branches and becomes complicated, which has motivated us to use bounding-spheres and ray-planes, rather than bounding-boxes and ray-frustums.

## 2.3 Procedures of RTRPS

Algorithm 1 shows the RTRPS processing pseudo code to render a frame. Each sphere takes the affine transformation as discussed in Sect. 2.1. Then, until a given reflection depth in the frame is reached<sup>†</sup>, ray-plane generation and intersec-

<sup>†</sup>For example, in the case of classic ray-tracing [29], the initial rays shot from the eye point to the screen pixels have intersection tests with the objects. If some of the initial rays actually hit reflective objects, the secondary rays representing the reflection are generated, where the original intersection points become the new ray origins. These processes are performed for each ray origin and repeated until the number of maximum reflection depth is reached.

**Algorithm 1** Frame Processing in RTRPS

---

```

1: procedure FRAMEPROCESSING
2:   For each sphere  $s$  in spherelist
3:     Perform affine transformation for sphere  $s$ .
4:   end
5:   for  $d \leftarrow 1, \text{maxdepth}$  do
6:     For each origin  $o$  in originlist of depth  $d$ 
7:        $\text{frustum} = \text{GENERATERAYPLANE}(o)$ 
8:       For each sphere  $s$  in spherelist
9:          $\text{INTERSECTION}(\text{frustum}, s)$ 
10:      end
11:    end
12:  end
13: end procedure

```

---

tion tests are repeatedly performed.

GENERATERAYPLANE() firstly creates the rays radiated from each ray origin. Note that, in the rendering process of high quality images, a number of rays must be shot from the ray origin because of super-sampling (not only for the primary ray from the eye point), as shown in Fig. 1. Secondary rays from a diffuse surface and sampling rays for light source (i.e. shadow-rays) are the examples of such super-sampling. As well as the primary rays from the eye point, these super-sampling rays also share a single origin. In RTRPS, assuming that all the rays sharing an origin are aligned to the uniform grid facing the origin as illustrated in Fig. 2, we can exploit another kind of coherency of rays. This assumption makes it possible to merge a series of rays into a plane in order to reduce the number of intersection tests. Then, GENERATERAYPLANE() groups a series of rays in a row into a ray-plane as shown in Fig. 2 (a). The generated set of ray-planes sharing an origin is called frustum (Fig. 2 (b)). Since the frustum is formed for a uniform grid, the generated frustum contains two sets of planes orthogonal to each other. We hereafter refer to these two sets as “row planes” and “column planes.” We do not have to distinguish which is row or column without losing generality. Although the grid-aligned rays may cause some artifacts in generated images, this kind of artifacts would not be significant as shown in the generated images in Sect. 4. Further evaluation of the effect of these artifacts is for future works.

For each frustum and sphere, INTERSECTION() performs intersection tests (Fig. 2 (c)). If it finds an intersection between the ray-planes and a sphere, the intersection tests between the ray, which is reproduced from the intersected ray-planes (Fig. 2 (d)), and the objects in the sphere are carried out, finally updating the frame buffer values. Because INTERSECTION() is repeatedly executed for each frustum and sphere, it is the most time-consuming function in RTRPS. The next section shows the techniques for making this function faster.

### 3. Acceleration Techniques for RTRPS

As shown in the previous section, the processing in a frame basically consists of three procedures: the affine transformation of spheres, GENERATERAYPLANE() and INTERSEC-

**Algorithm 2** Intersection Test in RTRPS

---

```

1: procedure INTERSECTION(frustum  $f$ , sphere  $s$ )
2:   For each plane  $rp$  in  $f.\text{rowPlanes}$ 
3:     Calculate the distance between  $rp$  and the center of  $s$ .
4:     If the distance is less than the radius of  $s$ , mark  $rp$  as hit.
5:   end
6:   For each plane  $cp$  in  $f.\text{columnPlanes}$ 
7:     Do the same as for  $rp$ .
8:   end
9:   For each sphere  $rp$  in  $f.\text{rowPlanes}$ 
10:    For each sphere  $cp$  in  $f.\text{columnPlanes}$ 
11:      if Both  $rp$  and  $cp$  are marked as hit then
12:        For each triangle  $tri$  in sphere  $s$ 
13:          Check intersection between  $tri$  and the ray  $r$ 
14:            reproduced from  $cp$  and  $rp$ 
15:        end
16:      end if
17:    end
18:  end
19: end procedure

```

---

TION(). Since, in these three procedures, INTERSECTION() is the most time-consuming, this section discusses five acceleration techniques for INTERSECTION(). In the rest of this paper, we assume that all the objects are represented as a set of triangles.

Algorithm 2 shows the detailed processing in INTERSECTION(), which takes a frustum  $f$  (including row planes and column planes) and sphere  $s$  as arguments. Firstly, it checks the intersection between all the row and column ray-planes in  $f$  and  $s$ . The distance between a plane and the center of a sphere is given by Eq. (1). If a pair of row and column planes has an intersection with  $s$ , it means that the ray shared by these planes can also intersect  $s$  and the intersection checks are triggered between the triangles contained in the sphere and that ray. We call hereafter this algorithm ‘baseline.’

Because INTERSECTION() is the most time-consuming in RTRPS, we apply the following five acceleration techniques to the baseline:

- For plane-sphere intersection
  - Parallel projection technique (Sect. 3.1.1)
  - Frustum culling technique (Sect. 3.1.2)
  - Binary-tree plane technique (Sect. 3.1.3)
- For ray-triangle intersection
  - Exploiting the shared ray origin (Sect. 3.2.2)
  - Exploiting the aligned rays (Sect. 3.2.3)

These techniques are explained in the following subsections in detail.

#### 3.1 Plane and Sphere Intersection

##### 3.1.1 Parallel Projection Technique

The baseline algorithm requires only one 3D vector inner product calculation to obtain the distance between the plane

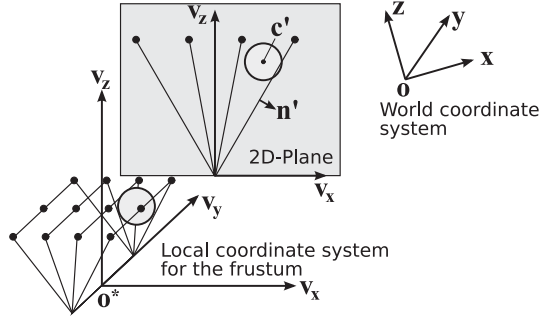


Fig. 4 Ray-planes and a bounding sphere on the plane of projection.

and sphere. Furthermore, we can reduce the number of operations in the distance calculation by using the parallel projection technique, which utilizes the property of row planes and column planes such that they share a common line of intersection.

This technique needs two modifications on the baseline algorithm. The first modification is on `GENERATERAYPLANE()`, where the set of ray-planes (i.e. frustum) is projected onto the plane orthogonal to all the concerning ray-planes, as shown in Fig. 4. Note that, for each frustum, two projected planes, i.e., the row- and column-projected planes exist. The second modification is on the distance calculation processed in `INTERSECTION()` lines 3 and 7. Using the 2D projected coordinate system, the distance between them can be obtained by only one 2D vector inner product calculation, instead of a 3D inner product, because it can be done by an intersection test between a line and a circle instead of a plane and a sphere.

In order to implement the parallel projection technique, all the planes and spheres must be affine-transformed into the local coordinate system ( $\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z, \mathbf{o}^*$ ) defined by the grid and origin of the rays. The transformation matrix  $\mathbf{T}_p$  is calculated in `GENERATERAYPLANE()` for each ray origin. The intersection test in the local coordinate system can be realized by the following steps:

- Step 1** Obtain the 2D normal vector  $\mathbf{n}'$  of the line (projected plane). This step is done in `GENERATERAYPLANE()`.
- Step 2** Given the center of sphere  $\mathbf{c}$  in the world coordinate, the center of an affine-transformed sphere in the local coordinate is obtained as  $\mathbf{c}' = \mathbf{T}_p \mathbf{c}$ . Depending on the projected plane, one of the three elements of  $\mathbf{c}'$  can be ignored in the plane. For example, in the case of Fig. 4, the element for  $\mathbf{v}_y$  can be ignored and the other two elements form the 2D position vector  $\mathbf{c}'' = (c'_x, c'_z)$ .
- Step 3** Calculate the distance of the line (projected plane) and the center of the circle given by the equation  $d = |\mathbf{n}' \cdot \mathbf{c}''|$ , because  $p_w$  in Eq. (1) becomes zero. If  $d < r$ , the plane intersects the sphere.

Steps 2 and 3 are performed in `INTERSECTION()`.

The parallel projection technique can reduce the number of calculations by using 2D inner products instead of 3D one but introduces some extra calculations such as obtaining  $\mathbf{T}_p$ ,  $\mathbf{n}'$ , and the affine transformation of  $\mathbf{c}$ . The cal-

culations for  $\mathbf{T}_p$  and  $\mathbf{n}'$  cause no problem, since they can be obtained in `GENERATERAYPLANE()` which is executed only once for each shared origin of rays. In contrast, the cost of affine transformation of  $\mathbf{c}$  may become a problem, because it is performed for every sphere and frustum.

In order to clarify the calculation cost of the parallel projection technique, we use the number of floating point operations as the metric. We assume that the costs of a floating-point addition and multiplication are the same and equivalent to one *flop* (floating-point operation). The cost for transformation of the center of a sphere is 18 flops per sphere. The costs for an intersection test based on 3D inner product and 2D inner product are six flops and three flops, respectively. Therefore, given the number of ray-planes  $n$  in a frustum, the computational cost for the plane-sphere intersections is given by:

$$PlaneSphereCost_{baseline} = 6n \text{ [flops]} \quad (2)$$

$$PlaneSphereCost_{projected} = 18 + 3n \text{ [flops]} \quad (3)$$

Equations (2) and (3) indicate that the projection technique is effective when the number of ray-planes in a frustum exceeds  $3 \times 3$ . When the primary rays (i.e. reflection depth is one) are shot, the number of ray-planes in the frustum is more than  $100 \times 100$ . For secondary rays,  $16 \times 16$  or more rays in common are radiated in distribution ray tracing and photon mapping. Hence, the parallel projection technique achieves great speedup on the baseline algorithm.

### 3.1.2 Frustum Culling Technique

The technique shown in Sect. 3.1.1 reduces the operations within one intersection. The frustum culling technique reduces the number of intersection tests. This technique is based on the fact that any sphere does not intersect with a ray-plane if the sphere is located at outside of the outermost plane in the frustum. The intersection tests for such a sphere are not necessary.

We explain the culling technique in the projected 2D space. Note that the planes forming the outline of a frustum become the two lines in the 2D space. Figure 5 shows all the cases of the relationship between a circle and outlines of a frustum. The frustum culling test is performed for the two outlines and a sphere by the following three steps.

- Step 1** For each of the outlines, determine whether the projected sphere (circle) intersects the projected plane (line) or not, and obtain the sign of the distance between them (left side of Fig. 6).
- Step 2** If the circle intersects either of the lines in Step 1, check whether or not the intersection point of the two outlines is encircled within the circle (right side of Fig. 6).
- Step 3** Based on the results of the tests in the above steps, determine the cases 1 to 13 in Table 1. In the cases of 5, 6, 7, 8, 9 and 10, the rest of intersection tests with the concerning sphere is unnecessary.



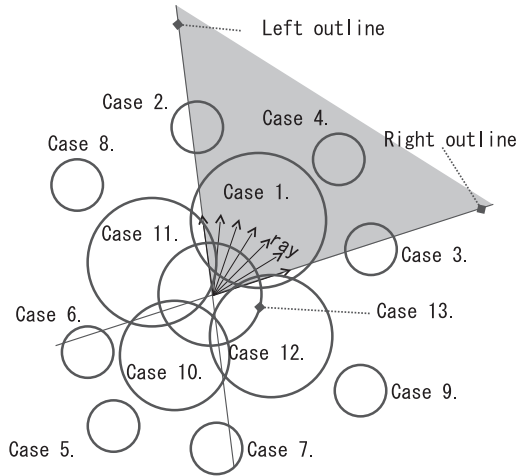


Fig. 5 Relative locations of sphere and a frustum for culling.

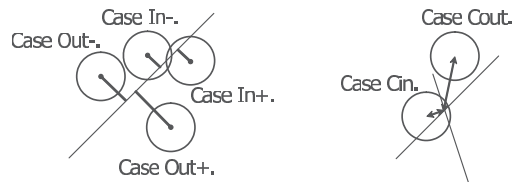


Fig. 6 Test conditions for frustum culling.

Table 1 Results of the frustum culling test (Only the shaded cases require actual intersection tests).

right plane \ left plane	Out-	In-		In+		Out+
Out-	8	2				4
In-	6	Cout	Cin	Cin	Cout	3
		11	13	13	1	
In+		Cout	Cin	Cin	Cout	
		10	13	13	12	
Out+	5	7				9

Using this technique, the redundant intersection tests for spheres can be skipped.

### 3.1.3 Binary-Tree Plane Technique

In most of the ray tracers, each of the rays traverses the object space and takes intersection tests by using a spatial data structure. In our RTRPS, on the other hand, a ray-plane does not traverse the object space directly. Instead, each sphere takes intersection tests with the ray-planes in a frustum as shown in Algorithm 1 lines 8-10, as if the *object group* (in a sphere) traversed among the ray-planes. Then we can use the last technique to reduce the number of intersection tests presented in this paper, 'binary-tree plane technique.' Based on the binary-tree structured order, the sphere takes intersection tests with column or row ray-planes. Figure 7 shows an example of such binary-tree structured ray-planes which have already been projected to a 2D-plane by the parallel projection technique. When an intersection test between a projected line and a circle is completed, the sign of the dis-

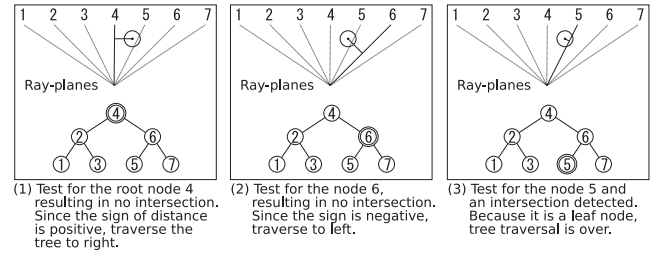


Fig. 7 An example of binary-tree plane technique.

tance between them shows their spatial relationship, which can be used to determine whether further intersection tests are necessary or not.

In this technique, all the ray-planes within a frustum are structured into a binary tree, such that the center plane (the 4th plane in the case of Fig. 7) is located on the root node. The intersection test of a sphere and a frustum begins at the root of the tree, and continues by traversing the tree. The direction of the traversal is determined by the intersection test result of the concerning plane and sphere, and the sign of their distance, using the following simple rules:

**Rule 1** When a plane intersects the sphere, both of the child nodes must be traversed.

**Rule 2** When it does not intersect, the sign of the distance between the plane and sphere is examined. If positive, traverse the tree to the right. Otherwise, traverse to the left.

As shown in Fig. 7, using this simple technique, more than half of the intersection tests of spheres and ray-planes can be skipped, if we adequately set the size of spheres and the density of the ray-planes. Especially, when the number of planes per frustum is large, this technique is very effective.

Besides, the set of row/column ray-planes sharing an origin can be implemented as an array, not a linked list, because they are built at once when the ray origin is given. Then, the identifiers of the ray-plane array are rearranged to form a binary tree. The same binary tree of identifiers can be reused anytime as long as the number of ray-planes in a frustum does not change. For instance, the number of ray-planes for the diffuse rays from the intersection point is usually given as a constant value at the beginning of rendering by the user. All the origins of diffuse rays can share a single binary tree, and therefore the cost for creating binary trees can be ignored.

### 3.2 Ray and Triangle Intersection

The following acceleration techniques are for the intersection calculation between rays and triangles. In the INTERSECTION() function of RTRPS, the intersection of ray-planes and a sphere is firstly checked and the intersected rays are reproduced from the intersected ray-planes. After that, the triangles in the sphere are examined if they intersect the reproduced rays. This procedure means that the rays share the

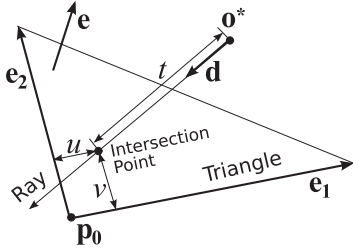


Fig. 8 Ray and triangle intersection test.

common origin, which can be exploited to accelerate the intersection tests. Furthermore, RTRPS assumes that all the rays are aligned to the grid in the local coordinate system, which leads us to another acceleration technique.

In this section, most commonly used intersection test technique presented by Möller [30] is briefly introduced, and our two accelerated techniques, “common origin technique” and “aligned ray-plane technique,” follow.

### 3.2.1 Möller’s Intersection Technique

Our idea originates from the ray-triangle intersection test proposed by Möller [30], which enhanced the original method [31]. As shown in Fig. 8, the normalized ray vector  $\mathbf{d}$  and the origin of the ray  $\mathbf{o}^*$  are given. The triangle is represented by the location vector  $\mathbf{p}_0$ , edge vectors  $\mathbf{e}_1$  and  $\mathbf{e}_2$ , and normal vector  $\mathbf{e} (= \mathbf{e}_1 \times \mathbf{e}_2)$ . The intersection point in the plane including the triangle can be represented by barycentric coordinates  $(u, v)$ , given by Eqs. (4a) to (4f).

$$\mathbf{p}'_0 = \mathbf{p}_0 - \mathbf{o}^* \quad (4a)$$

$$t' = \mathbf{p}'_0 \cdot \mathbf{e} \quad (4b)$$

$$w = \mathbf{d} \cdot \mathbf{e} \quad (4c)$$

$$\mathbf{g} = \mathbf{d} \times \mathbf{p}'_0 \quad (4d)$$

$$u' = -\mathbf{g} \cdot \mathbf{e}_2 \quad (4e)$$

$$v' = \mathbf{g} \cdot \mathbf{e}_1 \quad (4f)$$

where  $(t', u', v') = (tw, uw, vw)$ . If the following conditions are all met, the triangle does intersect the ray<sup>†</sup>.

$$u' \geq 0, v' \geq 0, \text{ and } u' + v' \leq w \quad (5)$$

This technique is well designed to begin the calculation of Eqs. (4a) and (4b) immediately after the ray origin is given. Especially in the case of the primary rays radiated from the eye point,  $\mathbf{p}'_0$  and  $t'$  for all the triangles can be calculated in advance, because all the rays share a single ray origin, i.e. the eye point. This pre-computed  $\mathbf{p}'_0$  and  $t'$  can be used for all the rays repeatedly, reducing the number of operations in INTERSECTION(), the most intensively executed function.

### 3.2.2 Common Origin Technique

In RTRPS, using the fact that all the rays in a frustum share the common origin, we can accelerate the original Möller

technique. This calculation procedure, which was described in our previous paper [32], is summarized below.

Using the rule of the scalar triple product, Eqs. (4e) and (4f) can be transformed respectively as follows:

$$u' = -\mathbf{g} \cdot \mathbf{e}_2 = -(\mathbf{d} \times \mathbf{p}'_0) \cdot \mathbf{e}_2 = \mathbf{d} \cdot (-\mathbf{p}'_0 \times \mathbf{e}_2) \quad (6a)$$

$$v' = \mathbf{g} \cdot \mathbf{e}_1 = (\mathbf{d} \times \mathbf{p}'_0) \cdot \mathbf{e}_1 = \mathbf{d} \cdot (\mathbf{p}'_0 \times \mathbf{e}_1) \quad (6b)$$

Then, the whole intersection procedure becomes as follows:

$$\mathbf{p}'_0 = \mathbf{p}_0 - \mathbf{o}^* \quad (4a \text{ again})$$

$$t' = \mathbf{p}'_0 \cdot \mathbf{e} \quad (4b \text{ again})$$

$$w = \mathbf{d} \cdot \mathbf{e} \quad (4c \text{ again})$$

$$\mathbf{h}_u = -\mathbf{p}'_0 \times \mathbf{e}_2 \quad (6c)$$

$$\mathbf{h}_v = \mathbf{p}'_0 \times \mathbf{e}_1 \quad (6d)$$

$$u' = \mathbf{d} \cdot \mathbf{h}_u \quad (6e)$$

$$v' = \mathbf{d} \cdot \mathbf{h}_v \quad (6f)$$

In addition to Eqs. (4a) and (4b) of the Möller technique, Eqs. (6c) and (6d) can also be processed as soon as the ray origin is provided. Although this technique can decrease the number of operations after  $\mathbf{d}$  is given, the total computation cost of the technique might be larger than the original technique, because the computation of  $\mathbf{h}_u$  and  $\mathbf{h}_v$  requires an excess vector product. We discuss the cost of the computation in Sect. 3.2.4

### 3.2.3 Aligned Ray-Plane Technique

In RTRPS, all the rays are shot to the uniform grid facing their origins. It means that all the rays contained in a ray-plane have the same gradient to an axis of the local coordinate system. We can use this characteristic for further acceleration of the intersection test.

Let us assume the local coordinate system  $(\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z, \mathbf{o}^*)$  shown in Fig. 9. The grid, to which the rays are radiated, is  $a_z$  away from  $\mathbf{o}^*$ . A ray is shot to a point  $(a_x, a_y)$  on the grid plane. Then, the ray direction vector  $\mathbf{d}$  can be expressed as follows:

$$\mathbf{d} = \frac{1}{l} (a_x \mathbf{v}_x + a_y \mathbf{v}_y + a_z \mathbf{v}_z) \quad (7)$$

$$l = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

From Eqs. (4c), (6e), (6f) and (7),  $u, v, w$  are described as

$$\begin{cases} u' &= \frac{1}{l} \{ (a_x \mathbf{v}_x + a_y \mathbf{v}_y + a_z \mathbf{v}_z) \cdot \mathbf{h}_u \} \\ v' &= \frac{1}{l} \{ (a_x \mathbf{v}_x + a_y \mathbf{v}_y + a_z \mathbf{v}_z) \cdot \mathbf{h}_v \} \\ w &= \frac{1}{l} \{ (a_x \mathbf{v}_x + a_y \mathbf{v}_y + a_z \mathbf{v}_z) \cdot \mathbf{e} \} \end{cases} \quad (8)$$

Here, let us see again the intersection condition given by

<sup>†</sup>  $t$  is used later to determine the closest triangle if multiple intersected triangles exist.

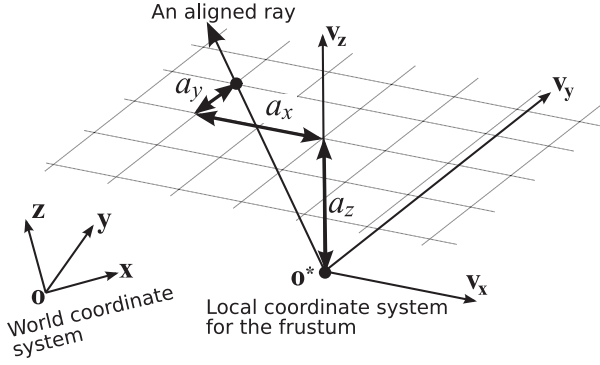


Fig. 9  $a_x, a_y, a_z$  of the aligned ray technique.

Eq. (5), where  $u'$  and  $v'$  are checked simply greater than or equal to zero, and the sum of  $u'$  and  $v'$  is compared with  $w$ . Therefore, the parameter  $(1/l)$  is not actually needed for the intersection test, and we can just remove it:

$$\begin{cases} u^* = a_x(\mathbf{v}_x \cdot \mathbf{h}_u) + a_y(\mathbf{v}_y \cdot \mathbf{h}_u) + a_z(\mathbf{v}_z \cdot \mathbf{h}_u) \\ v^* = a_x(\mathbf{v}_x \cdot \mathbf{h}_v) + a_y(\mathbf{v}_y \cdot \mathbf{h}_v) + a_z(\mathbf{v}_z \cdot \mathbf{h}_v) \\ w^* = a_x(\mathbf{v}_x \cdot \mathbf{e}) + a_y(\mathbf{v}_y \cdot \mathbf{e}) + a_z(\mathbf{v}_z \cdot \mathbf{e}) \end{cases} \quad (8')$$

In these equations, all the inner products can be calculated after the origin of the rays is fixed and the local coordinate parameters are given. Furthermore, since the grid plane can be given arbitrarily, we can set  $a_z = 1$ . Therefore, immediately after the origin of the rays is given, the third terms in Eq. (8') can be calculated for all the triangles. Similarly, all the rays forming a ray-plane share the same  $a_x$  or  $a_y$  depending on whether it is a row plane or column plane. Assuming that the  $a_y$  is shared by the rays on a ray-plane, the second term of Eq. (8') can be processed as soon as the ray-plane is given, and the addition of the second and third terms can also be completed. Finally, after a ray in the ray-plane is selected, only one multiplication and addition are enough for each of  $u^*, v^*, w^*$ .

### 3.2.4 Comparing the Techniques

The common origin and aligned ray-plane techniques are adopted to reduce the number of operations for individual rays. These techniques attempt to shift the shared operations among rays to the early phases of the process. However, in the case where only few rays share the operations, these techniques might degrade the performance.

Table 2 shows the number of floating-point operations per triangle for the three intersection techniques. In Möller technique and the common origin technique, the intersection test for a triangle is performed in two steps, immediately after the position of the ray origin is given, and after all the parameters of a ray are given. Since the common origin technique shifts more operations to the first stage, it can reduce the total number of floating-point operations when the triangle is used in the intersection tests repeatedly with many rays. In this case, the first stage operations can be skipped. The third technique, aligned ray-plane technique,

Table 2 Floating point operations required for each triangle intersection.

		Num. of FP op's at the origin notification	Num. of FP op's at the ray-plane notification	Num. of FP op's at the ray notification
Möller	Add	5	0	10
	Mul	3	0	17
	Total	8	0	27
Common origin	Add	11	0	7
	Mul	15	0	11
	Total	26	0	18
Aligned ray-plane	Add	29	3	3
	Mul	42	3	3
	Total	71	6	6

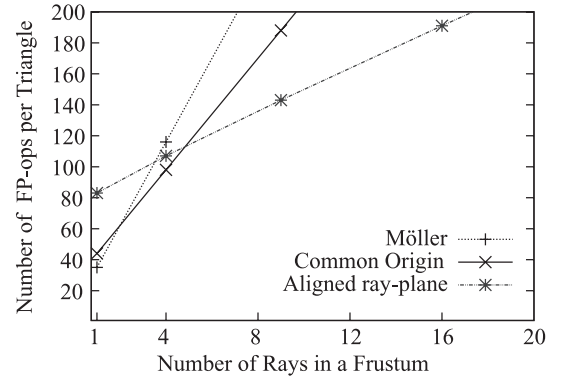


Fig. 10 Number of floating point operations required for intersection test between a frustum and triangle.

shifts further operations to the first stage and it has the intermediate stage executed when the row or column ray-plane is given. The more triangles are repeatedly used, the more operations can be reduced with the third technique.

Figure 10 illustrates the number of floating point operations required for the intersection test between a triangle and a frustum based on Table 2. The horizontal axis is the number of rays in a frustum. It obviously shows that if a frustum has more than  $2 \times 2$  rays, the aligned ray-plane technique reduces the number of operations drastically.



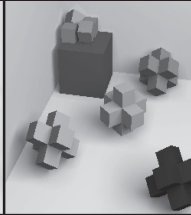
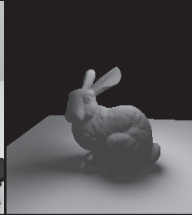
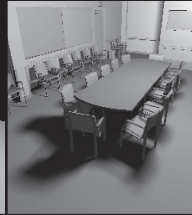
In RTRPS, the best calculation technique is selected dynamically for each frustum, according to the number of rays in the frustum based on the results shown in Fig. 10. For example, the primary rays shot to the screen from the eye point are merged into frustums with a large number of rays, resulting in the use of aligned ray-plane technique. However, if a ray intersects a specular surface, it produces a frustum with a single ray, resulting in the use of the classical Möller technique.

## 4. Performance Evaluations

We evaluate RTRPS from multiple viewpoints. Firstly, the potential of grouping rays radiated to a uniform grid plane is evaluated in the case of distribution ray tracing (DRT). In DRT, both the primary rays from eye point and the secondary rays from intersection points are used. Next, the effect of the frustum culling technique and the binary-tree planes technique is discussed for the DRT. Finally, the total effectiveness of RTRPS is estimated for FHRT, because most of the recent high speed ray tracers are optimized for



**Table 3** Scene conditions and generated images.

		Z33	Roses	Cubes	Bunny	Conference
Object information	Num. of bounding spheres	24794	2678	210	71093	193102
	Num. of leaf node spheres	23638	2612	177	69452	172264
	Max depth of hierarchy	3	3	3	3	3
	Num. of triangles	61828	11396	708	69455	344528
Images (generated by Distribution Ray Tracing)						

FHRT. Although RTRPS is not developed for FHRT but for high quality image generation using secondary rays, we will show that RTRPS achieves comparable performance to the FHRT-optimized ones. Especially, considering the time for the spatial structure update, which is mandatory for dynamic scene generation, RTRPS shows better performance than the FHRT-optimized counterparts even in the case of FHRT. Therefore, RTRPS is expected to be much more effective in high quality rendering of dynamic scenes which requires tracing the secondary rays.

#### 4.1 Experimental Environments

The ray tracing programs were written in Visual C++ .Net. They run on a commodity PC with a 2.4 GHz Intel Core 2 Duo E6600 processor and 2 GB DDR2 800 MHz SDRAM with MS Windows XP Professional SP2. To optimize the programs for X86 processors, we used Intel C++ Compiler Version 9.0, but note that SIMD and multi-thread operations have not fully been exploited, because they need extensive manual optimization. It means that there is further room of optimization of RTRPS code for X86 processors. We used single-precision floating-point calculations in the programs. All rendering time in this paper does not include shading time because we focus on the performance of intersection techniques in this paper.

Five test scenes shown in Table 3 were used. The spheres containing the triangles are built into a hierarchical structure in each scene. The depth of the hierarchy is three, and it is constructed by a simple script which makes intermediate nodes based on the structures in the scene graph, such as wheels, windows in the case of Z33 scene. Hence, if a rigid part, a door for example, moves, we just apply an affine transformation to the top-level or intermediate sphere including the moving part. As shown in the table, each of the leaf node sphere has, on average, only several triangles in it, which means that the number of intersection tests between planes and spheres has a significant effect on the total rendering time.

Also note that no optimization for the sphere hierarchy has been made. Hence, further performance gain will be obtained by optimizing the sphere hierarchy.

Table 4 shows the number of rays generated in distri-

**Table 4** Number of generated rays for distribution ray tracing.

		Z33	Roses	Cubes	Bunny	Conference
Total number of rays		14039552	33619968	28803328	8082176	16634880
Primary ray	Num. of rays	65536	65536	65536	65536	65536
	Resolution	256x256	256x256	256x256	256x256	256x256
Shadow ray	Num. of rays	13974016	33554432	16646144	8016640	16569344
	Num. of square lights	1	2	1	1	1
	Sampling resolution	16x16	16x16	16x16	16x16	16x16
Diffuse reflection ray	Num. of rays	0	0	12091648	0	0
	Secondary ray depth	0	0	1	0	0
	Sampling resolution	-	-	16x16	-	-

bution ray tracing for the five scenes. We used a packet including 32 planes equivalent to a  $16 \times 16$  ray-packet, which means that RTRPS can reduce the number of ray-bounding-box or plane-sphere intersection tests to 1/8 compared to the conventional ray-packet methods using the same packet size. Because distribution ray tracing requires a huge number of secondary rays, primary rays are less than 1 % of the total number rays for all the scenes in Table 4.

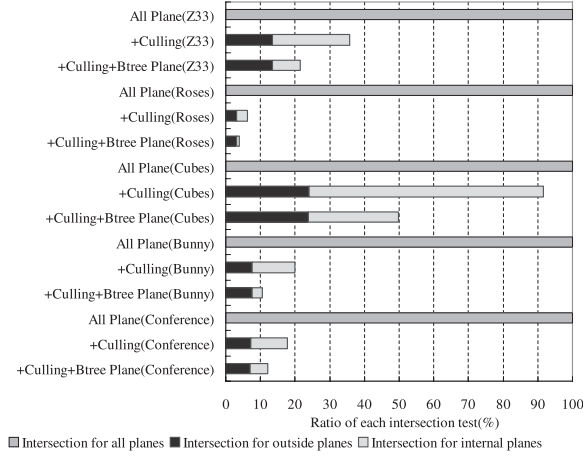
#### 4.2 Grouping Rays into a Plane

Merging rays into a packet or frustum has been reported effective for fast ray tracing [18], [33]. The speedup has been achieved by two reasons. The first reason is exploiting the uniformity of ray processing in a frustum which well matches the SIMD functionality of modern processors. The second reason is high cache hit ratio caused by the locality, i.e., the rays in a single frustum will likely need the intersection tests with almost the same set of triangles. RTRPS takes these two advantages. In addition, RTRPS uses the plane-sphere intersection concept discussed in Sect. 2.2 to reduce the number of intersection tests. Because a leaf node sphere has only several triangles, the number of plane-sphere intersection tests dominates the total rendering time.

Table 5 shows the speedup of frame rendering by RTRPS compared to the case of “random rays.” “Random rays” means that neither of ray-plane, frustum culling nor binary-tree plane technique is used. This table shows that RTRPS achieves 13 to 392 times speedup, depending on the scenes, over the random ray case. Note that, the number of the ray-triangle intersections is the same for both RTRPS and random rays, because both use the same sphere hierarchy and generated rays. Therefore, the speedup is ob-

**Table 5** Execution time for distribution ray tracing (excluding ray generation time).

	Z33	Roses	Cubes	Bunny	Conference
Random rays without frustum (sec)	230.0	385.9	321.9	4822.5	143684.5
RTRPS with frustum (sec)	17.0	2.0	3.8	69.1	365.7
Speedup	x13.5	x190.1	x84.7	x69.8	x392.9

**Fig. 11** Breakdown of the effects of the frustum culling and the binary-tree planes.

tained by reducing the number of plane-sphere intersection tests and acceleration techniques such as frustum culling and binary-tree planes.

### 4.3 Frustum Culling and Binary-Tree Planes

Figure 11 shows the normalized number of intersection tests between the planes and spheres in RTRPS. For each scene, “All Plane” means the case which does not use the frustum culling or the binary-tree plane techniques. The *outside* intersection means the intersection tests between the outermost planes of a frustum (i.e. left and right outlines in Fig. 5) and a sphere, which is inevitable even by using the frustum culling technique. The *internal* intersection means the intersection tests between the other planes and a sphere. As we can see, these two techniques can eliminate up to 95% of plane-sphere intersection tests. However, the effect of the two techniques is depending on the scene property.

For the Roses scene, most of the triangles are located at the center of the image, and the triangles forming the flowers and leaves are very small. Considering that the number of triangles in a leaf node sphere is three or four, the size of leaf node spheres is also quite small. In this case, most of the intersection tests between ray-planes and spheres can be skipped by using the frustum culling technique alone.

Cubes has different characteristics. Even if the number of triangles and spheres is small, their solid angle is relatively larger than that of Roses. Larger triangles enlarge the solid angle of the leaf node sphere, making the frustum culling less effective. Because of the same reason, frustum

culling is not effective for the diffuse rays radiated in the range of approximately 180 degrees. As shown in Fig. 11, only less than 10% of the ray-planes can skip the intersection tests. However, even in this scene, the *internal* intersection tests can be reduced significantly by the binary-tree plane technique.

For the other three scenes, the two techniques work effectively and can avoid 80% to 90% of intersection tests.

### 4.4 Applying to the First-Hit Ray Tracing

In this section, RTRPS is compared with famous ray tracing methods [19], [25], [26], [34] and a GPU ray tracer constructing BVH rapidly [27]. The RTRPS is designed for high quality image generation using a large number of secondary rays, such as diffuse rays, at an interactive rate. Then, we want to compare the RTRPS with the ray tracers based on a similar concept. However, most of the recent competitive ray tracers are optimized for FHRT, which forces us to evaluate the RTRPS in the FHRT field. We chose the rendering time and update time between two frames as the performance metric, because they are commonly used in the conventional methods<sup>†</sup>.

Table 6 summarizes the performance from the articles and that of our RTRPS.

MLRTA [19] and MLRTUG [34] are the fast methods using a kd-tree and a uniform grid, respectively, as the spatial data structure. Recently, Zhou et al. have achieved a fast kd-tree update by using GPU (GeForce 8800 ULTRA), which requires less than 60 msec for scenes with over 100 K triangles [28]. On the other hand, RTRPS uses the bounding spheres for a shorter update time. Conference is used as the test scene because it was commonly used in [19] and [34]. We can see that the spheres can drastically reduce the update time, resulting in an equivalent total time<sup>††</sup>.

RTRPS is also compared to the proposal based on the axis aligned bounding box (AABB) hierarchies [25], where two AABB techniques using surface area heuristic (SAH) and median split (Med), respectively, were implemented. From the table, we can see that RTRPS shows better performance than AABB with SAH. For AABB-Med, although the total time in the table of RTRPS is shorter, it is not easy to compare them directly, because AABB-Med had smaller image size than RTRPS, but it radiated shadow and reflection rays. However, we can say that RTRPS achieves higher or comparative performance to the AABB-based methods.

Next, let us compare RTRPS with the fastest FHRT

<sup>†</sup>Unfortunately, it is not easy to precisely compare two rendering systems, because 1) complete source code is not always published, 2) the computational conditions (CPU, memory, compiler, optimization technique and OS) may not be the same, 3) the scene condition (the eye point, angle view, and so on) cannot be the same even if the scene data is the same. Hence, we chose the test scenes commonly used in the recent articles and attempted to keep the scene condition to be the same as the articles to the best of our ability.

<sup>††</sup>The update time of MLRTA was adduced in [5]. The update time of MLRTUG is derived from [9].

**Table 6** Performance comparison with related works.

Name	Data structure	Scene	Rendering time	Update time	Total time	Secondary ray condition	Image size	Hardware
MLRTA [19]	KD-tree	Conference	51 msec	1410 msec	1461 msec	None	1024x1024	Pen4 3.2 GHz
MLRTUG [34]	Uniform grid	Conference	250 msec	89 msec	339 msec	None (with shading)	1024x1024	Pen4 3.2 GHz
RTRPS	Sphere	Conference	405 msec	2 msec	407 msec	None	1024x1024	Core2Duo 2.4 GHz
RT-DEFORM [25]	AABB(Med)	Bunny	-	23 msec	166 msec	Shadow and reflection rays	512x512	Pen4-Dual 2.8 GHz
	AABB(SAH)	Bunny	178 msec	23 msec	201 msec	None	1024x1024	Pen4-Dual 2.8 GHz
RTRPS	Sphere	Bunny	124 msec	0.52 msec	125 msec	None	1024x1024	Core2Duo 2.4 GHz
Frustum+BVH [26]	BVH	Conference	108 msec	-	-	None (with shading)	1024x1024	Opteron 2.6 GHz
LBVH [27]	BVH	Conference	149 msec	19 msec	-	None	1024x1024	GeForce 280 GTX
RTRPS	Sphere	Conference	405 msec	2 msec	407 msec	None	1024x1024	Core2Duo 2.4 GHz

proposal using bounding volume hierarchies (BVH) reported in [26]. Because of its careful optimization for the SIMD functionality, the BVH method achieves the four times shorter rendering time than RTRPS. Although the update time of BVH was not reported in [26], we can approximately estimate that it would be the same as or longer than that of AABB techniques. Taking account of the number of triangles in Conference which is four or more times larger than Bunny, the update time of BVH in Conference should be longer than 100 msec. In order to shorten the update time of BVH, Lauterbach et al. have introduced LBVH (Linear Bounding Volume Hierarchy) and its GPU processing [27]. They have achieved the update of Linear BVH structure in 19 msec per frame with Geforce 280 GTX for Conference scene. It is still slower than RTRPS but means that the bounding volume (boxes and spheres) strategy is suitable for GPU implementation than the other data structure. Since there is a lot of room for optimizing RTRPS to the GPU, SIMD, and multi-thread functionality, RTRPS is still competitive in the total time with the BVH method even in the field of FHRT.

## 5. Concluding Remarks and Future Work

In this paper, we have proposed a new ray tracing scheme named RTRPS, which uses bounding spheres, instead of well-known bounding boxes or kd-trees, as the spatial data structure. The spheres as the spatial data structure can reduce the cost of reconstructing the data structure between frames in a dynamic scene rendering, because only the translation of the sphere is sufficient even if the movement of an object includes rotation. In addition, RTRPS groups the rays into a set of ray-planes, which makes it possible to exploit unused inter-ray-group coherency. The combination of ray-planes and the spheres can decrease the cost of a ray-sphere intersection test and the number of intersection tests.

Three acceleration techniques, parallel projection, frustum culling, and binary-tree planes, further reduce the number of intersection tests for ray-planes and the spheres. For the intersection tests between rays and triangles, which are the finest level of the intersection test, RTRPS dynamically chooses the fastest one from the three intersection techniques, classical Möller, the common origin technique, and the aligned ray-plane technique.

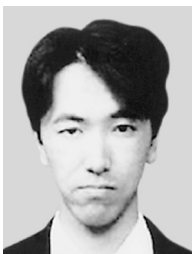
The performance evaluations have shown that RTRPS can achieve 390 times speedup for distribution ray tracing.

The evaluations have also shown that RTRPS gives competitive performance even in the field of first hit ray tracing.

## References

- [1] J.D. MacDonald and K.S. Booth, "Heuristics for ray tracing using space subdivision," *Visual Computer*, vol.6, no.3, pp.153–166, 1990.
- [2] M.J. Charney and I.D. Scherson, "Efficient traversal of well-behaved hierarchical trees of extents for ray-tracing complex scenes," *Visual Computer*, vol.6, no.3, pp.167–178, 1990.
- [3] V. Havran, *Heuristic Ray Shooting Algorithms*, PhD thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, 2000.
- [4] I. Wald and V. Havran, "On building kd-trees for ray tracing, and on doing that in  $O(n \log n)$ ," *RT '06: Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pp.61–70, 2006.
- [5] S. Popov, J. Gunther, H.P. Seidel, and P. Slusallek, "Experiences with streaming construction of SAH kd-trees," *RT '06: Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pp.89–94, 2006.
- [6] W. Hunt, W.R. Mark, and G. Stoll, "Fast kd-tree construction with an adaptive error bounded heuristic," *RT '06: Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pp.81–88, 2006.
- [7] A. Fujimoto, T. Tanaka, and K. Iwata, "Accelerated ray-tracing system," *IEEE Comput. Graph. Appl.*, vol.6, no.4, pp.16–26, 1986.
- [8] J.G. Cleary and G. Wyvill, "Analysis of an algorithm for fast ray tracing using uniform space subdivision," *Visual Computer*, vol.4, no.2, pp.65–83, 1988.
- [9] T. Ize, C.R.I. Wald, and S.G. Parker, "An evaluation of parallel grid construction for ray tracing dynamic scenes," *RT '06: Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pp.47–54, 2006.
- [10] K.Y. Whang, J.W. Song, J.W. Chang, J.Y. Kim, W.S. Cho, C.M. Park, and I.Y. Song, "Octree-r: An adaptive octree for efficient ray tracing," *IEEE Trans. Vis. Comput. Graphics*, vol.01, no.4, pp.343–349, 1995.
- [11] G. van den Bergen, "Efficient collision detection of complex deformable models using aabb trees," *J. Graphics Tools*, vol.2, no.4, pp.1–13, 1997.
- [12] S. Gottschalk, M.C. Lin, and D. Manocha, "OBTree: A hierarchical structure for rapid interference detection," *SIGGRAPH '96: Proc. 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pp.171–180, 1996.
- [13] P.S. Heckbert and P. Hanrahan, "Beam tracing polygonal objects," *SIGGRAPH '84: Proc. 11th Annual Conference on Computer Graphics and Interactive Techniques*, pp.119–127, 1984.
- [14] J. Amanatides, "Ray tracing with cones," *SIGGRAPH '84: Proc. 11th Annual Conference on Computer Graphics and Interactive Techniques*, pp.129–135, 1984.
- [15] M. Shinya, T. Takahashi, and S. Naito, "Principles and applications of pencil tracing," *SIGGRAPH '87: Proc. 14th Annual Conference on Computer Graphics and Interactive Techniques*, pp.45–54, 1987.
- [16] J. Genetti, D. Gordon, and G. Williams, "Adaptive supersampling in object space using pyramidal rays," *Comput. Graph. Forum*, vol.17, no.1, pp.29–54, March 1998.

- [17] S. Teller and J. Alex, "Frustum casting for progressive interactive rendering," Tech. Rep. MIT/LCS/TR-740, 1998.
- [18] I. Wald, P. Slusallek, C. Benthin, and M. Wagner, "Interactive rendering with coherent ray tracing," *Comput. Graph. Forum (Proc. Eurographics 2001)*, vol.20, no.3, pp.153–164, 2001.
- [19] A. Reshetov, A. Soupikov, and J. Hurley, "Multi-level ray tracing algorithm," *ACM Trans. Graphics (TOG)*, vol.24, no.3, pp.1176–1185, 2005.
- [20] V. Havran, R. Herzog, and H.P. Seidel, "On fast construction of spatial hierarchies for ray tracing," *RT '06: Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pp.71–80, 2006.
- [21] S.M. Rubin and T. Whitted, "A 3-dimensional representation for fast rendering of complex scenes," *SIGGRAPH '80: Proc. 7th Annual Conference on Computer Graphics and Interactive Techniques*, pp.110–116, 1980.
- [22] H. Weghorst, G. Hooper, and D.P. Greenberg, "Improved computational methods for ray tracing," *ACM Trans. Graphics (TOG)*, vol.3, no.1, pp.52–69, 1984.
- [23] J. Goldsmith and J. Salmon, "Automatic creation of object hierarchies for ray tracing," *IEEE Comput. Graph. Appl.*, vol.7, no.5, pp.14–20, 1987.
- [24] T.L. Kay and J.T. Kajiya, "Ray tracing complex scenes," *SIGGRAPH'86: Proc. 13th Annual Conference on Computer Graphics and Interactive Techniques*, pp.269–278, 1986.
- [25] C. Lauterbach, S.E. Yoon, D. Tuft, and D. Manocha, "RT-DEFORM: Interactive ray tracing of dynamic scenes using bvhs," *RT '06: Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pp.39–46, 2006.
- [26] I. Wald, S. Boulos, and P. Shirley, "Ray tracing deformable scenes using dynamic bounding volume hierarchies," *ACM Trans. Graphics (TOG)*, vol.26, no.1, p.6, 2007.
- [27] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, "Fast BVH construction on GPUs," *Comput. Graph. Forum (Proc. Eurographics 2009)*, vol.28, no.2, pp.375–384, 2009.
- [28] K. Zhou, Q. Hou, R. Wang, and B. Guo, "Real-time kd-tree construction on graphics hardware," *ACM Trans. Graphics (TOG)*, vol.27, no.5, pp.1–11, 2008.
- [29] T. Whitted, "An improved illumination model for shaded display," *Commun. ACM*, vol.23, no.6, pp.343–349, 1980.
- [30] *Graphics Tools — The jgt editors' choice*, A.K. Peters, Ltd, 2005.
- [31] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," *J. Graphics Tools*, vol.2, no.1, pp.21–28, 1997.
- [32] K. Komatsu, Y. Kaeriyama, K. Suzuki, H. Kobayashi, and T. Nakamura, "Intersection algorithm for ray tracing using packets," *Information Technology Letters*, pp.265–268, 2007.
- [33] S. Boulos, D. Edwards, J.D. Lacewell, J. Kniss, J. Kautz, I. Wald, and P. Shirley, "Packet-based whitted and distribution ray tracing," *GI '07: Proc. Graphics Interface 2007*, 2007.
- [34] I. Wald, T. Ize, A. Kensler, A. Knoll, and S.G. Parker, "Ray tracing animated scenes using coherent grid traversal," *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pp.485–493, 2006.



**Ken-ichi Suzuki** received the B.E degree, Master degree on information sciences, and Ph.D. from Tohoku University in 1992, 1994, 1997, respectively. He worked for Miyagi National College of Technology as an assistant professor from 1997 to 2003. He was an assistant professor at Graduate School of Information Sciences, Tohoku University from 2003 to 2008. From 2008, he is an associate professor at Department of Information and Communication Engineering, Tohoku Institute of Technology.



**Yoshiyuki Kaeriyama** is currently an Engineer of Nikon Corporation. His research interests include computer graphics and its hardware implementation. He received the B.E. in Mechanical Engineering, and the M.S. in Information Sciences and the Ph.D. degrees from Tohoku University in 1999, 2001, and 2007 respectively.



**Kazuhiko Komatsu** is currently a Post-Doctoral Fellow in Cyberscience Center, Tohoku University. His research interests include computer graphics and high-performance computing. He received the B.E. Degree in Mechanical Engineering, and the M.S. Degree in Information Sciences and Ph.D. from Tohoku University in 2002, 2004, and 2008 respectively.



**Ryusuke Egawa** received the B.E degree, Master degree on information sciences from Hiroaki University in 1999, 2001, respectively, and the Ph.D. on information sciences from Tohoku University 2004. He is an assistant professor at Graduate School of Information Sciences, Tohoku University from 2005.



**Nobuyuki Ohba** received a Ph.D. from the Department of Electrical and Communication Engineering in the Graduate School of Engineering at Tohoku University in 1986. He joined the Japan Science Institute of IBM Japan, Ltd. in 1986, and is an advisory researcher of IBM Research, Tokyo Research Laboratory, IBM Japan, Ltd. His interests include computer architecture, VLSI design, computer graphics, and wireless communications.



**Hiroaki Kobayashi** is currently Director and Professor of Cyberscience Center and Professor of the Graduate School of Information Sciences, Tohoku University. His research interests include high-performance computer architectures, grid and P2P computing, and multimedia applications. He received the B.E. Degree in Communication Engineering, and the M.E. and D.E. Degrees in Information Engineering from Tohoku University in 1983, 1985, and 1988 respectively. He is a senior member of IEEE CS, and a member of ACM and IPSJ.