IPSJ Transactions on Advanced Computing Systems Vol. 1 No. 1 85–95 (June 2008)

Regular Paper

A Fast Ray Frustum-Triangle Intersection Algorithm with Precomputation and Early Termination

Kazuhiko Komatsu,[†1] Yoshiyuki Kaeriyama,[†1]
Kenichi Suzuki,[†1] Hiroyuki Takizawa[†1]
and Hiroaki Kobayashi[†1]

Although ray tracing is the best approach to high-quality image synthesis, much time is required to generate images due to its huge amount of computation. In particular, ray-primitive intersection tests still dominate the execution time required for ray tracing, and faster ray-primitive intersection algorithms are strongly required to interactively generate higher-quality images with more advanced effects. This paper presents a new fast algorithm for the intersection tests that makes a good use of ray and object coherence in ray tracing. The proposed algorithm utilizes the features whereby the rays in a bundle share the same origin and have massive coherence. By reducing the redundant calculations in the innermost intersection tests for the bundles by precomputation and early termination, the proposed algorithm accelerates the intersection tests. Experimental results show that the proposed algorithm achieves 1.43 times faster intersection tests compared with Möller's algorithm by exploiting the features of the bundles of rays.

1. Introduction

Recently, there has been a growing demand for high-quality images in various fields such as entertainment, industrial designs, and environmental assessments based on lighting simulation. In these fields, image generation methods based on the global illumination model play an important role. One of the most basic methods of the global illumination model is ray tracing proposed in 1980s[1)–3)]. Although ray tracing can generate photo realistic images, the huge computational cost has made an interactive image generation very difficult.

Ray tracing is accelerated by reducing the number of intersection tests using the spatial data structure such as Bounding Volume Hierarchy (BVH)[4)], Kd-tree[5),6)], and uniform grid[7)]. In order to reduce the number of intersection tests efficiently, a complicated building process for a sophisticated data structure is essential. The sophisticated data structure requires a so large memory that it is difficult to deal with large scenes for ray tracing. Thus, a fast intersection algorithm using the simple data structure is required to deal with large scenes.

To improve the speed, the *ray packet* approach has been proposed[8)]. Bundling two or more rays into a ray packet, it exploits the coherence of adjacent rays. Ray packets usually consist of a group of some primary rays, shadow rays, or diffuse rays as shown in **Fig. 1**. A ray packet in which rays share the same origin is especially called a *ray frustum*. These rays in a ray frustum have a large coherence because most of them are very similar and need to perform similar calculations. For instance, they tend to traverse similar spaces and intersect with the same objects. By using the SIMD processing of modern CPUs, the intersection tests can quickly be performed[9),10)]. In addition, by extra simple operations using the feature of the ray frustums, the number of the intersection tests between ray frustums and triangles can be reduced[6),11)–13)].

However, the demand for faster intersection tests is still growing for higher-quality images. Although ray tracing for simple visibility tests can be interactively performed, ray tracing for surface light sources, multi-level reflec-

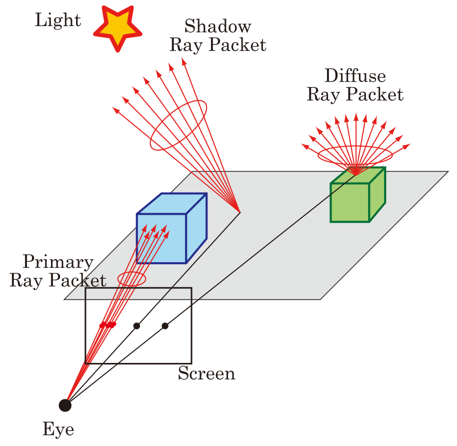


Fig. 1 Various ray packets.

†1 Tohoku University

tions/refractions, and indirect lighting needs a speedup of intersection tests. Since the fast intersection test can also quicken the traversal of the spatial data structure [14], the acceleration of the intersection test is a quite effective approach to achieve an interactive ray tracing.

This paper proposes a new intersection algorithm for ray frustum-triangle intersection tests by using precomputation and an early termination. The proposed algorithm focuses on the feature whereby the rays in a ray frustum share the same origin. As a result, the number of intersection operations can be reduced. In addition, by optimization to the modern processor architecture, the proposed algorithm can achieve high-speed intersection tests.

This paper is organized as follows. Section 2 reviews related work. In Section 3, we propose a new intersection algorithm for ray frustum-triangle intersection tests. In Section 4, we evaluate the performance of the proposed algorithm through experiments. Section 5 presents concluding remarks and future work.

2. Related Work

Many intersection algorithms have been proposed in the field of computer graphics. The projection algorithm [9], Möller's algorithm [15], and Pluecker's algorithm [16] have been used for recent ray tracers.

Basically, a ray-triangle intersection test is performed in the following steps. Suppose the intersection test between a ray $R(t) = O + tD$ and a triangle of vertices P_0, P_1, and P_2 as shown in **Fig. 2**. Then, the intersection is tested by calculating the point H at which the ray R intersects with the plane defined by the vertices P_0, P_1, and P_2. Only if the intersection point H is located within the triangle, the ray R intersects with the triangle.

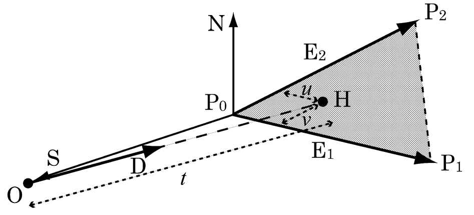


Fig. 2 A ray and a triangle.

2.1 Projection Intersection Algorithm

The projection algorithm is an optimized method of the barycentric coordinates test [17],[18]. Since the intersection test is performed by projecting rays and triangles into an axis-aligned plane instead of the 3D coordinate system, the projection algorithm requires less computation than that of the barycentric test.

In the original projection algorithm, the edges of a triangle E_1, E_2, the normal N, and the projected coordinate are recalculated for every intersection test. However, the calculated results are always the same when the triangle is unchanged. By precomputing and storing these values, hence, the redundant recalculation can be avoided. Although a larger memory capacity is required to store the precomputed results, the intersection tests can quickly be performed. In addition, the exploitation of the locality of reference to the precomputed data leads to the effective utilization of the large cache and the avoidance of expensive cache misses. By calculating multiple rays together instead of calculating each ray individually, the intersection tests are further accelerated by using SIMD operations.

The projection algorithm is suitable for the ray-triangle intersection test. However, it is not optimized for the ray frustum-triangle intersection test; more calculations can be avoided by the precomputation using the features of a ray frustum as discussed later.

2.2 Möller's Intersection Algorithm

In Möller's algorithm [15], from a ray R and a point $H(u, v) = (1 - u - v)P_0 + uP_1 + vP_2$ on a triangle, the intersection point between the ray and the triangle is obtained by:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(D \times E_2) \cdot E_1} \begin{pmatrix} (S \times E_1) \cdot E_2 \\ (D \times E_2) \cdot S \\ (S \times E_1) \cdot D \end{pmatrix}, \qquad (1)$$

where $\times$ denotes a cross product, $\cdot$ denotes a standard dot product, $S = O - P_0$, $E_1 = P_1 - P_0$, and $E_2 = P_2 - P_0$. Firstly, $(D \times E_2) \cdot E_1$ is calculated and tests whether the direction of a triangle faces the ray origin. Then, u and v are calculated and check whether they meet $0 \leq u$, $0 \leq v$, and $u + v \leq 1$. Finally, the distance t is calculated. In this intersection algorithm, the edges and the normal of a triangle are recalculated for every intersection test as in the case of the original projection algorithm.

Möller's algorithm has been improved by eliminating the redundant calculations[19]. The scalar triple product rules and the commutative property of the cross product are applied to Eq. (1). Equation (1) is rewritten as follows:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(E_2 \times E_1) \cdot D} \begin{pmatrix} (E_1 \times E_2) \cdot S \\ (S \times D) \cdot E_2 \\ (D \times S) \cdot E_1 \end{pmatrix} = -\frac{1}{N \cdot D} \begin{pmatrix} N \cdot S \\ -(D \times S) \cdot E_2 \\ (D \times S) \cdot E_1 \end{pmatrix}, \tag{2}$$

where $N = E_1 \times E_2$. Since the edges E_1, E_2, and the normal N are constant, these values can be precomputed before intersection calculations. The improved Möller's algorithm can be expressed in the following manner, where $u' = -(D \times S) \cdot E_2$ and $v' = (D \times S) \cdot E_1$:

```
1: function PRECOMPUTATION FOR A TRIANGLE ($P_0, P_1, P_2$)
2:     $E_1 \leftarrow P_1 - P_0$
3:     $E_2 \leftarrow P_2 - P_0$
4:     $N \leftarrow E_1 \times E_2$
5:     return($E_1, E_2, N$)
6: end function
7: function MÖLLER INTERSECTION TEST ($D, O, N, P_0, E_1, E_2$)
8:     $f_1 \leftarrow N \cdot D$
9:     if $f_1 \geq 0$ then
10:        return($NOTHIT$)                    ▷ Early Termination.
11:    end if
12:    $S \leftarrow O - P_0$
13:    $G \leftarrow D \times S$
14:    $u' \leftarrow -G \cdot E_2$
15:    if $u' < 0$ then
16:        return($NOTHIT$)                    ▷ Early Termination.
17:    end if
18:    $v' \leftarrow G \cdot E_1$
19:    if $v' < 0$ then
20:        return($NOTHIT$)                    ▷ Early Termination.
21:    end if
22:    if $u' + v' > -f_1$ then
23:        return($NOTHIT$)                    ▷ Early Termination.
24:    end if
25:    $f_2 \leftarrow N \cdot S$
26:    $t \leftarrow -f_2 \ / \ f_1$
27:    return($HIT, u', v', t$)
28: end function
```

The precomputation enables faster intersection tests than the original Möller intersection tests. As in the case of the projection algorithm, however, it is not optimized for the ray frustum.

2.3 Pluecker's Intersection Algorithm

Pluecker's algorithm[20),21)] can quickly perform the intersection tests. A directed line in the 3D coordinate system can be represented in the 6D Pluecker coordinate system[22),23)]. Given two points A and B in the 3D coordinate system, the line L through the two points is defined as $L = [A - B, A \times B]$. A ray R can be denoted by $R = [D, D \times O]$.

The Pluecker inner product gives the positional relationship between two lines. The inner product is defined as $L_0 * L_1 = U_0 \cdot V_1 + U_1 \cdot V_0$ for two lines $L_0 = [U_0, U_1]$ and $L_1 = [V_0, V_1]$ in the coordinate system. If the inner product equals to zero, the two lines intersect as shown in **Fig. 3** (b); if the inner product does not equal

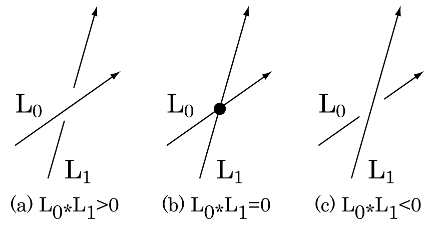


Fig. 3 Pluecker Line-Line test.

to zero, the two lines do not intersect as shown in Fig. 3 (a) and Fig. 3 (c). By taking the inner products of a ray and the three edges of a triangle, the ray-triangle intersection can be tested. If all the inner products between the ray and the three edges have the same sign, the ray intersects with the triangle.

Carsten has improved Pluecker's algorithm for ray frustums [16]. Using the feature whereby all rays within a ray frustum share a common origin, the inner product between the ray R and the triangle edge T is simplified to $R * T' = D \cdot ((A - O) \times (B - O))$ by translating the shared ray origin to the origin of the coordinate system, where T' indicates the translated edge. Since $(A - O) \times (B - O)$ is precomputable immediately after the ray origin has been decided, the intersection tests between a ray in a frustum and a triangle can be performed by calculating three dot products. The algorithm of the improved Pluecker intersection test is described as follows:

```
function PRECOMPUTATION FOR A TRIANGLE ($P_0, P_1, P_2$)
    $E_1 \leftarrow P_1 - P_0$
    $E_2 \leftarrow P_2 - P_0$
    $N \leftarrow E_1 \times E_2$
    return($N$)
end function
function PRECOMPUTATION FOR A RAY FRUSTUM ($O, N, P_0, P_1, P_2$)
    $T_0 \leftarrow (P_1 - O) \times (P_0 - O)$
    $T_1 \leftarrow (P_2 - O) \times (P_1 - O)$
    $T_2 \leftarrow (P_0 - O) \times (P_2 - O)$
    $f_2 \leftarrow N \cdot (O - P_0)$
    return($T_0, T_1, T_2, f_2$)
end function
function PLUECKER INTERSECTION TEST ($D, N, T_0, T_1, T_2, f_2$)
    $f_1 \leftarrow N \cdot D$
    if $f_1 \geq 0$ then
        return($NOTHIT$)                    ▷ Early Termination.
    end if
    $\alpha \leftarrow D \cdot T_0$
    $\beta \leftarrow D \cdot T_1$
    $\gamma \leftarrow D \cdot T_2$
    if $\alpha, \beta, \gamma$ have not the same sign then
        return($NOTHIT$)                    ▷ Early Termination.
    end if
    $t \leftarrow -f_2 \ / \ f_1$
    return($HIT, t$)
end function
```

Note that Pluecker's algorithm needs four dot products in total, because it also requires one more dot product to check the face of a triangle.

The improved Pluecker algorithm can achieve the fast ray frustum-triangle intersection test. However, it requires a large memory capacity and the higher cost of precomputation.

3. Novel Intersection Algorithm for Ray Frustum

We propose a new ray frustum-triangle intersection algorithm based on Möller's algorithm. The intersection calculation time can be further reduced by incorporating the ray frustum-triangle intersection tests into Möller's algorithm.

By precomputing all rays in a ray frustum that are shot from the same origin point, a large amount of calculation can be removed. In other words, $N \cdot S$ in Eq. (2) is constant for every ray included in a frustum and can be calculated in advance. In addition, more matrix elements can be precomputed by applying the scalar triple product rules to Eq. (2) again. Eventually, Eq. (2) is rewritten as follows:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = -\frac{1}{N \cdot D} \begin{pmatrix} N \cdot S \\ D \cdot (-S \times E_2) \\ D \cdot (S \times E_1) \end{pmatrix}. \tag{3}$$

Since $N \cdot S$, $-S \times E_2$, and $S \times E_1$ of a triangle are constant for every ray within a single ray-frustum, these precomputed values can be used for all intersection tests of those rays. The proposed algorithm can be described as the following algorithm, where $u' = D \cdot (-S \times E_2)$ and $v' = D \cdot (S \times E_1)$:

```
1: **function** PRECOMPUTATION FOR A TRIANGLE $(P_0, P_1, P_2)$
2:     $E_1 \leftarrow P_1 - P_0$
3:     $E_2 \leftarrow P_2 - P_0$
4:     $N \leftarrow E_1 \times E_2$
5:     **return**$(E_1, E_2, N)$
6: **end function**
7: **function** PRECOMPUTATION FOR A RAY FRUSTUM $(O, N, P_0, E_1, E_2)$
8:     $S \leftarrow O - P_0$
9:     $G_u \leftarrow -S \times E_2$
10:    $G_v \leftarrow S \times E_1$
11:    $f_2 \leftarrow N \cdot S$
12:    **return**$(G_u, G_v, f_2)$
13: **end function**
14: **function** PROPOSED INTERSECTION TEST $(D, N, G_u, G_v, f_2)$
15:    $f_1 \leftarrow N \cdot D$
16:    **if** $f_1 \geq 0$ **then**
17:        **return**$(NOTHIT)$                    ▷ Early Termination.
18:    **end if**
19:    $u' \leftarrow D \cdot G_u$
20:    **if** $u' < 0$ **then**
21:        **return**$(NOTHIT)$                    ▷ Early Termination.
22:    **end if**
23:    $v' \leftarrow D \cdot G_v$
24:    **if** $v' < 0$ **then**
25:        **return**$(NOTHIT)$                    ▷ Early Termination.
26:    **end if**
27:    **if** $u' + v' > -f_1$ **then**
28:        **return**$(NOTHIT)$                    ▷ Early Termination.
29:    **end if**
30:    $t \leftarrow -f_2 \ / \ f_1$
31:    **return**$(HIT, u', v', t)$
32: **end function**
```

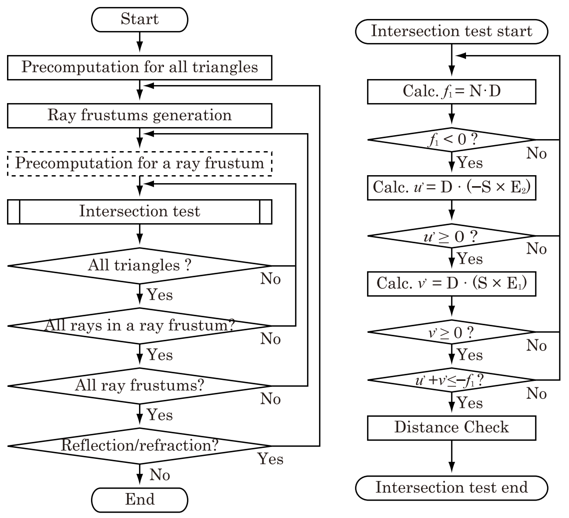


Fig. 4 Flowchart of the proposed ray frustum-triangle intersection test.

Figure 4 shows the flowchart of the proposed algorithm. The newly-introduced precomputation process is shown as the dashed box. The intersection test between a ray in a frustum and a triangle requires only three dot products at the maximum.

In every iteration, the proposed algorithm requires less dot products and less cross products than the other algorithms reviewed in Section 2. Compared to Möller's algorithms, it can eliminate one dot product and one cross product in each iteration.

In the case where all intersection tests of a ray are terminated by the first conditional check at line 16, the precomputed values are never used and hence the precomputation is redundant. In practice, however, the precomputed values are almost always reused many times, resulting in a significant reduction in the intersection calculation time.

Table 1 shows the number of floating-point operations of precomputation for a ray frustum to one triangle, an intersection test between one ray and one triangle, and intersection tests between rays in one frustum and one triangle, where F is the number of rays in a frustum. In this table, an early termination is not considered

Table 1 Number of operations of the precomputation for a frustum and the intersection test. F is the number of rays in a frustum.

Algorithm	Precomp.	Test	Total
Möller	0	34	$34F$
Projection	0	22	$22F$
Pluecker	41	21	$41+21F$
Proposed	26	17	$26+17F$

for comparing the algorithms in terms of the maximum amounts of computation. If F is 6 or more, the proposed algorithm needs the fewest operations for an intersection test. The greater the number of rays becomes, the faster the proposed algorithm can test intersections because of the precomputation.

The number of floating-point operations required by the proposed algorithm for the precomputation is less than that required by Pluecker's algorithm. While Pluecker's algorithm needs 41 floating-point operations for its precomputation, the proposed algorithm only needs 26 operations. The number of precomputed values used in the proposed algorithm is less than those of Möller's algorithm and Pluecker's algorithm during intersection tests. In the case of using single precision floating-point values, the data size required by each algorithm for the intersection tests involved in a frustum is as follows. Möller's algorithm, the projection algorithm, and Pluecker's algorithm need 48 bytes, 40 bytes, and 52 bytes, respectively. The proposed algorithm needs 40 bytes, which includes the x, y, and z components of vectors S, G_u, and G_v and the scalar value f_2. Thus, the proposed algorithm requires less memory capacity than do Möller's algorithm and Pluecker's algorithm.

Compared with the projection algorithm and Pluecker's algorithm, the proposed algorithm can effectively use the early termination as well as Möller's algorithm. If any conditional check does not succeed, a ray does not intersect with the current triangle, and is tested for the next triangle. In the projection algorithm and Pluecker's algorithm, several conditional checks are performed at the last stage of the intersection calculation. Thus, they have to perform a lot of floating-point operations before an early termination.

On the other hand, conditional checks in the proposed algorithm and Möller's algorithm are performed earlier than those in the projection algorithm and those in Pluecker's algorithm. As a result, the proposed algorithm and Möller's algorithm can skip more floating-point operations than can the projection and Pluecker's algorithm. For instance, while the second check of Pluecker's algorithm is performed after four dot products, the second check of the proposed algorithm and of Möller's algorithm is carried out after only two dot products. This effective early termination is an advantage of the proposed algorithm and Möller's algorithm over the projection algorithm and Pluecker's algorithm.

Besides, the proposed algorithm can exploit the potential of modern CPUs with large caches and powerful SIMD instructions. The data arrangement and alignment of the precomputed values lead to an effective utilization of the cache and an avoidance of expensive cache misses. Data parallel processing with SIMD instructions can further accelerate the intersection test by calculating multiple rays together instead of each ray individually.

4. Performance Evaluation

In order to evaluate the proposed algorithm, we did experiments on a PC equipped with a 2 GB DDR memory and an Intel Pentium 4 processor running at 3.4 GHz. In the experiments, Intel C++ Compiler v9.1 was used. The experiments were conducted by generating images from the Stanford data archives [24] shown in **Fig. 5**. All these images were generated by 2×2 ray frustums of primary rays.

Figure 6 and **Table 2** show the intersection calculation times of the proposed algorithm and the conventional algorithms. The resolution of the test images is 256×256 pixels. All the intersection algorithms are implemented by using Intel SSE intrinsics, which are C/C++ function-style macros. In order to evaluate the performance of the intersection algorithm itself, no spatial data structure is used. In the figure, the x-axis indicates the test scenes. The y-axis indicates the speedup ratio of each algorithm against Möller's algorithm.

This figure shows that the proposed algorithm achieves faster intersection tests than the other algorithms. For all the test scenes, the proposed algorithm can perform intersection tests faster than the others. Depending on the arrangement of triangles in the scene, the speedup ratio of the proposed algorithm against Möller's algorithm varies from about $1.1\times$ to $1.3\times$.

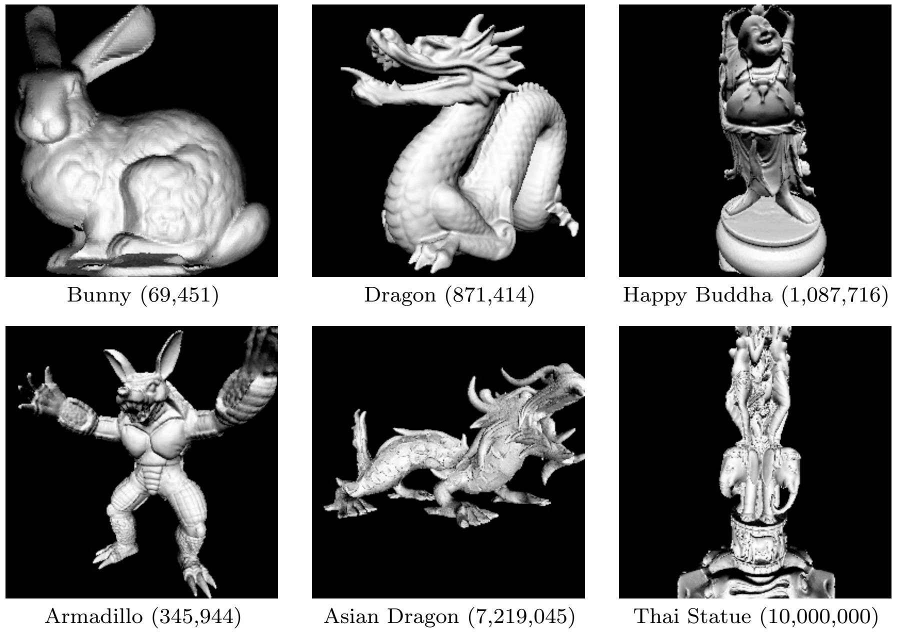


Fig. 5 Test scenes (Number of triangles).

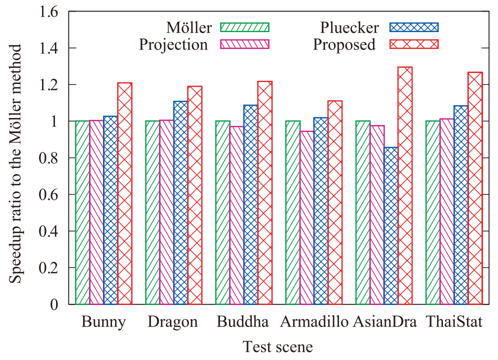


Fig. 6 Comparison of the intersection time.

Table 2 Intersection time (sec.).

Method	Bunny	Dragon	Happy Buddha
Möller	36.8	589.9	731.4
Projection	36.7	587.2	754.2
Pluecker	35.9	532.8	673.2
Proposed	30.5	496.0	601.3
Method	**Armadillo**	**Asian Dra.**	**Thai Stat.**
Möller	256.6	4,130.7	6,632.8
Projection	271.9	4,236.7	6,557.1
Pluecker	252.0	4,826.7	6,123.7
Proposed	231.1	3,191.5	5,240.8

Table 3 Breakdown of the average ratio of early termination points.

Term.	Möller	Projection	Pluecker	Proposed
1st	20.9%	29.6%	20.9%	20.9%
2nd	37.9%	34.2%	N/A	37.9%
3rd	27.8%	24.3%	N/A	27.8%
last	13.5%	11.9%	79.1%	13.5%

One of the reasons of the speedups is that the proposed algorithm requires less operations than do the others because of the precomputation.

Another reason is that the early termination works effectively in the proposed algorithm. The superiority of the proposed algorithm over Pluecker's algorithm is mainly due to the efficient early termination. It can skip the operations after an early termination point, and proceed to the next intersection iteration.

Table 3 shows the breakdown of early termination points in the six test scenes, where the percentage is the ratio of the number of early-terminated tests at each termination point to the total number of early-terminated tests. The experimental conditions are the same as those in Fig. 6. Note that the termination points of the projection algorithm are different from those of the other algorithms. This table shows that 86.5% (= 20.9%+37.9%+27.8%) of intersection tests in the proposed algorithm and Möller's algorithm are early-terminated, while only 20.9% of tests are early-terminated in Pluecker's algorithm. As a result, Pluecker's algorithm takes a longer calculation time than the proposed algorithm and Möller's algorithm.

In order to clarify the effects of the precomputation and the early termination of the proposed algorithm, **Fig. 7** compares the intersection times under the

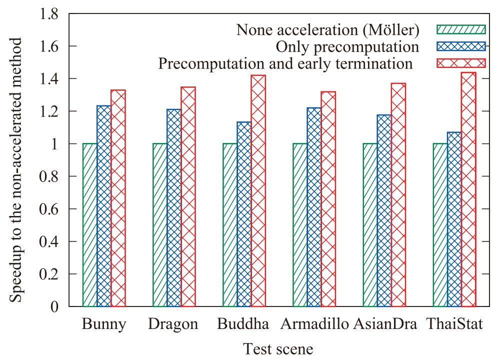


Fig. 7 Effects of precomputation and early termination.

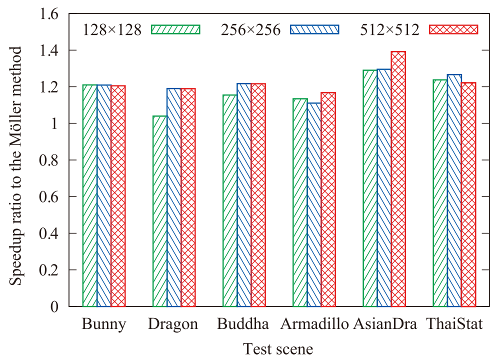


Fig. 8 Speedup of intersection time for various resolution.

three conditions; Möller's algorithm without the early termination, the proposed algorithm without the early termination, and the proposed algorithm with both the precomputation and the early termination. The y-axis indicates the speedup ratios of the intersection time to that of Möller's algorithm without the early termination. The other experimental conditions are the same as Fig. 6.

This figure shows that both the precomputation and the early termination contribute to the acceleration of intersection tests. The speedup by the precomputation is about 7 to 23%. The speedup by both the precomputation and the early termination is about 31 to 43%. The performance gain by the early termination varies greatly according to the test scenes, because the termination point depends on the arrangement of triangles in the scene.

To evaluate the performance when generating images with a different magnitude of coherence among rays, the following experiments were conducted under various image resolutions. A high resolution means that the rays in a frustum concentrate into a narrow region, resulting in a higher coherence. The other experimental conditions are the same as in Fig. 6. **Figure 8** shows the speedup ratios in generating various resolution images of each test scene. The y-axis indicates the speedup ratio of the intersection time of the proposed algorithm against that of Möller's algorithm.

This figure shows that the speedup is basically higher when the resolution is higher. As the resolution increases, the coherence of rays in a ray frustum also increases. This higher coherence leads to the performance improvement of the proposed intersection test.

The speedup of Bunny does not depend on the resolution because the coherence of the rays is high enough. It is confirmed that the breakdown of early-terminated points for Bunny of 128×128, 256×256, and 512×512 pixels are almost the same. It means that all rays in a frustum hit the same triangles even in the case of a 128×128-pixel image. Thus, there is no performance difference among the various resolution settings in Bunny.

For all the scenes, a certain speedup is achieved even if the resolution is 128×128, where rays are the least coherent in the experiments. This implies that the proposed algorithm is useful even for low coherence rays.

In order to evaluate the performance of the proposed algorithm with a spatial data structure, the experiments were conducted by applying SAH-BVH (Surface Area Heuristic-Bounding Volume Hierarchies) data structure [4)] to Möller's algorithm and the proposed algorithm. SAH-BVH is one of the most widely used data structures for the reduction of the number of intersection tests during an image generation.

Figure 9 and **Table 4** show the intersection time of Möller's algorithm and the proposed algorithm. The intersection time includes the traverse time of the SAH-BVH and the intersection test time of ray frustums and triangles. In this figure, the y-axis indicates the speedup ratio of the intersection time which includes the

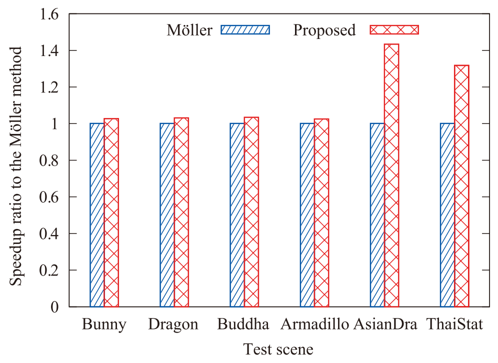


Fig. 9 Speedup of intersection time with SAH-BVH data structure.

Table 4 Intersection time with SAH-BVH traversal time (sec.).

Method	Bunny	Dragon	Happy Buddha
Möller	0.155	0.150	0.186
Proposed	0.150	0.146	0.180
Method	Armadillo	Asian Dra.	Thai Stat.
Möller	0.106	5.26	33.6
Proposed	0.104	3.67	25.6

traverse time of the SAH-BVH and the intersection test time of ray frustums and triangles. The maximum depths of the SAH-BVHs for Asian Dragon and Thai Statue are set to 12 and 10 respectively because of the limited memory capacity. The resolution of the test images is 256×256 pixels. All intersection algorithms are implemented by the standard C language without Intel SSE intrinsics and optimized for SIMD operations by Intel C++ Compiler optimizer.

SAH-BVH selects only triangles with high intersection probabilities and allows them to be tested by the intersection algorithms. As a result, intersection tests for such triangles are not often terminated earlier, and the performance gain by the early termination in Fig. 9 is less than that in Fig. 6. This means that the precomputation becomes more significant in Fig. 9.

The speedups become more remarkable for large scenes of many triangles such as Asian Dragon and Thai Statue. On the other hand, for small scenes, there are small differences in performance between Möller's algorithm and the proposed algorithm because only a small number of triangles selected by SAH-BVH are tested by those algorithms. In addition, as SAH-BVH is built based on the cost of testing an intersection, the use of a faster intersection algorithm leads to a more sophisticated data structure with less hierarchy levels. As a result, a faster intersection test can accelerate both intersection tests and a traversal in the data structure. Accordingly, the speedups of the proposed algorithm against Möller's algorithm in Fig. 9 are larger than those in Fig. 6.

5. Conclusions

In this paper, we have proposed a new fast intersection algorithm for ray frustum-triangle intersection tests. The proposed algorithm utilizes the features of rays with the same origin in a frustum. It eliminates redundant calculations by the proposed precomputation and the early termination boosts the innermost intersection test iteration.

We have implemented and evaluated the proposed algorithm. The experimental results have demonstrated the performance improvement of the proposed intersection algorithm. The proposed algorithm is up to 1.43 times faster than Möller's algorithm.

Our future work includes the application of the proposed algorithm not only to primary ray packets but also to shadow ray packets and diffuse ray packets. The proposed algorithm is expected to be useful even for those ray packets, because they are also coherent enough if quite a number of their sampling rays sharing the same origin are tested. In addition, the implementation of the proposed algorithm onto the GPU will appear in our future work.

Acknowledgments The authors would like to thank the anonymous reviews for valuable comments. The authors would like to thank Professor Emeritus Tadao Nakamura and Professor Ryusuke Egawa of Tohoku University for their comments for this research.

References

1) Whitted, T.: An improved illumination model for shaded display, *Comm. ACM*, Vol.23, No.6, pp.343–349 (1980).
2) Cook, R.L., Porter, T. and Carpenter, L.: Distributed ray tracing, *SIGGRAPH '84:*

Proc. 11th Annual Conference on Computer Graphics and Interactive Techniques, pp.137–145 (1984).

3) Kajiya, J.T.: The rendering equation, *SIGGRAPH '86: Proc. 13th Annual Conference on Computer Graphics and Interactive Techniques*, pp.143–150 (1986).

4) Wald, I., Boulos, S. and Shirley, P.: Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies, *ACM Trans. Graphics* (*Proc. ACM SIGGRAPH*) (2006).

5) Havran, V.: Heuristic Ray Shooting Algorithms, PhD Thesis, Czech Technical University (2000).

6) Reshetov, A., Soupikov, A. and Hurley, J.: Multi-Level Ray Tracing Algorithm, *SIGGRAPH '05: Proc. 32nd International Conference on Computer Graphics and Interactive Techniques*, 3, Vol.24, pp.1176–1185 (2005).

7) Wald, I., Ize, T., Kensler, A., Knoll, A. and Parker, S.G.: Ray Tracing Animated Scenes using Coherent Grid Traversal, *ACM Trans. Graphics* (*Proc. ACM SIGGRAPH*) (2006).

8) Wald, I., Slusallek, P., Benthin, C. and Wagner, M.: Interactive Rendering with Coherent Ray Tracing, *Computer Graphics Forum* (*Proc. Eurographics*), 3, Vol.20, pp.153–164 (2001).

9) Wald, I.: Realtime Ray Tracing and Interactive Global Illumination, PhD Thesis, Saarland University (2004).

10) Benthin, C., Wald, I., Scherbaum, M. and Friedrich, H.: Ray Tracing on the CELL Processor, *RT '06: IEEE Symposium on Interactive Ray Tracing* (2006).

11) Dmitriev, K., Havran, V. and Seidel, H.-P.: Faster Ray Tracing with SIMD Shaft Culling, *Research report MPI-I-2004-4-006* (2004).

12) Reshetov, A.: Faster Ray Packets-Triangle Intersection through Vertex Culling, *IEEE/EG Symposium on Interactive Ray Tracing* (2007).

13) Lauterbach, C., Chandak, A. and Manocha, D.: Interactive Sound Rendering in Complex and Dynamic Scenes using Frustum Tracing, *IEEE Trans. Visualization and Computer Graphics*, Vol.13, No.6 (2007).

14) Komatsu, K., Kaeriyama, Y., Zaitsu, D., Suzuki, K., Ohba, N. and Nakamura, T.: Packet-Primitive Intersection Method, *RT '06: Poster Compendium of IEEE Symposium on Interactive Ray Tracing*, p.6 (2006).

15) Möller, T. and Trumbore, B.: Fast, minimum storage ray triangle intersection, *Graphics Tools*, Vol.2, No.1, pp.21–28 (1997).

16) Benthin, C.: Realtime Ray Tracing on Current CPU Architectures, PhD Thesis, Saarland University (2006).

17) Badouel, D.: An Efficient Ray Polygon Intersection, *Graphics Gems*, pp.390–393 (1990).

18) Glassner, A.: *An Introduction to Ray Tracing*, ISBN: 0-12286-160-4, Morgan Kaufmann (1989).

19) Barzel, R.(ed.): *Graphics Tools—the jgt editors' choice*, A K Peters, Ltd. (2005).

20) Erickson, J.: Pluecker Coordinates, Ray Tracing News (1997). http://www.acm.org/tog/resources/RTNews/html/rtnv10n3.html#art11

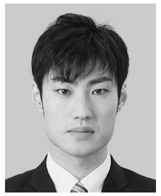
21) Shoemake, K.: Pluecker Coordinate Tutorial, Ray Tracing News (1998). http://www.acm.org/tog/resources/RTNews/html/rtnv11n1.html#art3

22) Sommerville, D.: *Analytical Geometry of Three dimensions*, Cambridge University Press (1939).

23) Teller, S.J.: Computing the antipenumbra of an area light source, *SIGGRAPH Computer Graphics*, Vol.26, No.2, pp.139–148 (1992).

24) Stanford Computer Graphics Laboratory: *The Stanford 3D Scanning Repository.* http://www-graphics.stanford.edu/data/3Dscanrep/

(Received October 9, 2007)
(Accepted March 11, 2008)



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

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

Kenichi 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.

Hiroyuki Takizawa is currently a senior assistant professor in the Graduate School of Information Sciences, Tohoku University. His research interests include high-performance computing systems and their applications, such as computer graphics. He received the bachelor's degree in Mechanical Engineering, and the master's and doctor's degrees in Information Sciences from Tohoku University in 1995, 1997 and 1999, respectively. He is a member of the IEEE CS, the IEICE, and the IPSJ.

Hiroaki Kobayashi is currently a Director and Professor of Cyberscience Center and a Professor of Graduate School of Information Sciences, Tohoku University. His research interests include high-performance computer architectures and their 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, IEICE and IPSJ.