LI-004

Accelerating Möller Intersection Algorithm Using Ray Packets

Kazuhiko Komatsu* Yoshiyuki Kaeriyama* Kenichi Suzuki* Hiroaki Kobayashi* Tadao Nakamura*

Abstract

Many implementation methods of ray tracing have been proposed, however, execution time of ray-primitive intersection tests still dominate the total execution of rendering, and faster algorithms have been strongly required. This paper presents a new fast algorithm for the intersection tests between packets of rays and triangles. Experimental results show that the proposed algorithm achieves faster intersection tests by exploiting the feature of the packets of rays.

keywords: ray tracing, intersection algorithm, ray packet, ray bundle

1 Introduction

A basic ray tracing algorithm was founded in 1980s, and since then many implementation methods have been proposed so far. Although ray tracing can generate photo realistic images, the huge computation of ray tracing makes quick image generation very difficult. The most computation comes mainly from the intersection tests between rays and objects. Therefore, it is essential to boost their processing speed in order to achieve interactive ray tracing.

Recently, the SIMD operations have attracted a lot of attention and have been an indispensable factor of fast ray tracing. Although traditional ray tracers process each ray independently, recent ray tracers have processed a set of coherent rays together because they tend to need similar calculations. A set of coherent rays is called a ray packet, which usually consists of a group of some primary rays, shadow rays, or diffuse rays as shown in Figure 1. Effectively treating the ray packets leads to the efficient use of SIMD operations and exploitation of memory reference locality[1]. However, the SIMD intersection algorithms between ray packets and objects have not fully been established.

In this paper, we propose a new intersection algorithm for packet-triangle intersection tests. By exploiting the special property of ray packets, it can perform fast intersection tests.

This paper is organized as follows. Section 2 shows related work. In Section 3, after briefly reviewing the base intersection algorithm, this paper proposes a new packet-triangle intersection algorithm. In Section 4, we evaluate the performance of the proposed algorithm through experimental results. Section 5 gives concluding remarks and future work.

*Graduate School of Information Sciences, Tohoku University

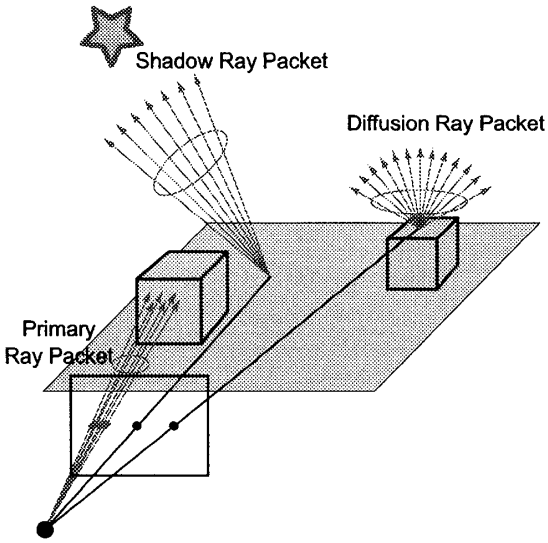


Figure 1: Various ray packets

2 Related Work

A lot of ray-triangle intersection algorithms have been proposed in the field of computer graphics. The projection intersection test and the Pluecker intersection test have been used for recent ray tracers[1][2].

In the original projection test, which is based on the barycentric coordinates test[3], the edges and the normal of a triangle and the plane to be projected are recalculated for every intersection test with this triangle although these values are always the same. The projection intersection test has been improved by reducing these redundant recalculations and exploiting the computational power of current CPU resources.

The Pluecker intersection test[4] quickly performs the most inside loop of the intersection tests. The Plucker inner product represents the position between two lines. The Pluecker 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 Pluecker coordinates, where $\cdot$ denotes a standard dot product. If the Pluecker inner product equals to zero, two lines intersect; otherwise, the two lines do not intersect and the relation either front side or back side is shown by its sign. By taking the Pluecker inner products of a ray and the three edges of a triangle, the ray-triangle intersection can be tested. If all the Pluecker inner products between the ray and the three edges have the same sign, the ray intersects the triangle.

The Pluecker intersection test has been improved by using ray packets[2]. It reduces intersection calculation

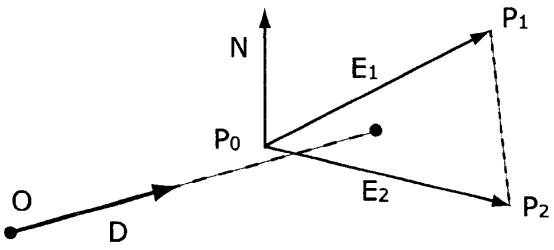


Figure 2: A ray and a triangle

by translating the origin of a ray packet to the origin of the coordinates. Only three dot instructions are required in the most inside loop of the intersection tests.

Although the projection test is suitable for the ray-triangle intersection test, it is not optimized for the packet-triangle intersection test because it does not exploit the property of the packet at all. The Pluecker test is widely used for the packet-triangle intersection test, however, the intersection time is not enough short to the interactive ray tracing. We focus on another intersection algorithm and optimize it for the packet-triangle in order to achieve faster packet-triangle intersection tests.

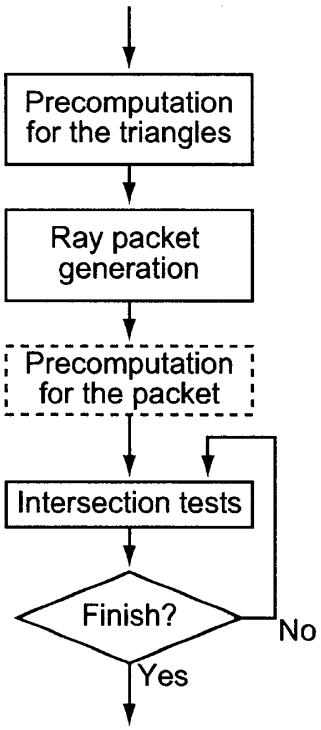


Figure 3: Flowchart of proposed packet-triangle intersection test

3 Intersection Algorithm

3.1 Möller-Trumbore Intersection Test

The Möller-Trumbore intersection test is one of the most widely used methods for fast ray-triangle intersection algorithm[5]. Consider a ray R going from the origin O to the direction D and a triangle with vertexes P_0, P_1, and P_2 shown in Figure 2. From a ray $R(t) = O + tD$ and a point $T(u,v) = (1-u-v)P_0 + uP_1 + vP_2$ on a triangle, the solution of the hit point between the ray and the triangle is obtained by:

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

where $S = O - P_0$, $N = E_1 \times E_2$, $E_1 = P_1 - P_0$, and $E_2 = P_2 - P_0$. In implementation, $(D \times E_2) \cdot E_1$ is first calculated and tested whether the triangle is front side or not. Then, u and v are calculated and checked whether they meets $0 \leq u$, $0 \leq v$, and $u + v \leq 1$ respectively. Finally, the distance t is calculated. In this intersection test, the edges and the normal of a triangle are recalculated for every intersection test as well as the original projection test.

The Möller-Trumbore intersection algorithm has been improved by omitting the redundant calculations [6]. In order to omit the calculation of the edges and the normal, the scalar triple product rules are applied to Equation (1). Equation (1) is represented as follows:

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

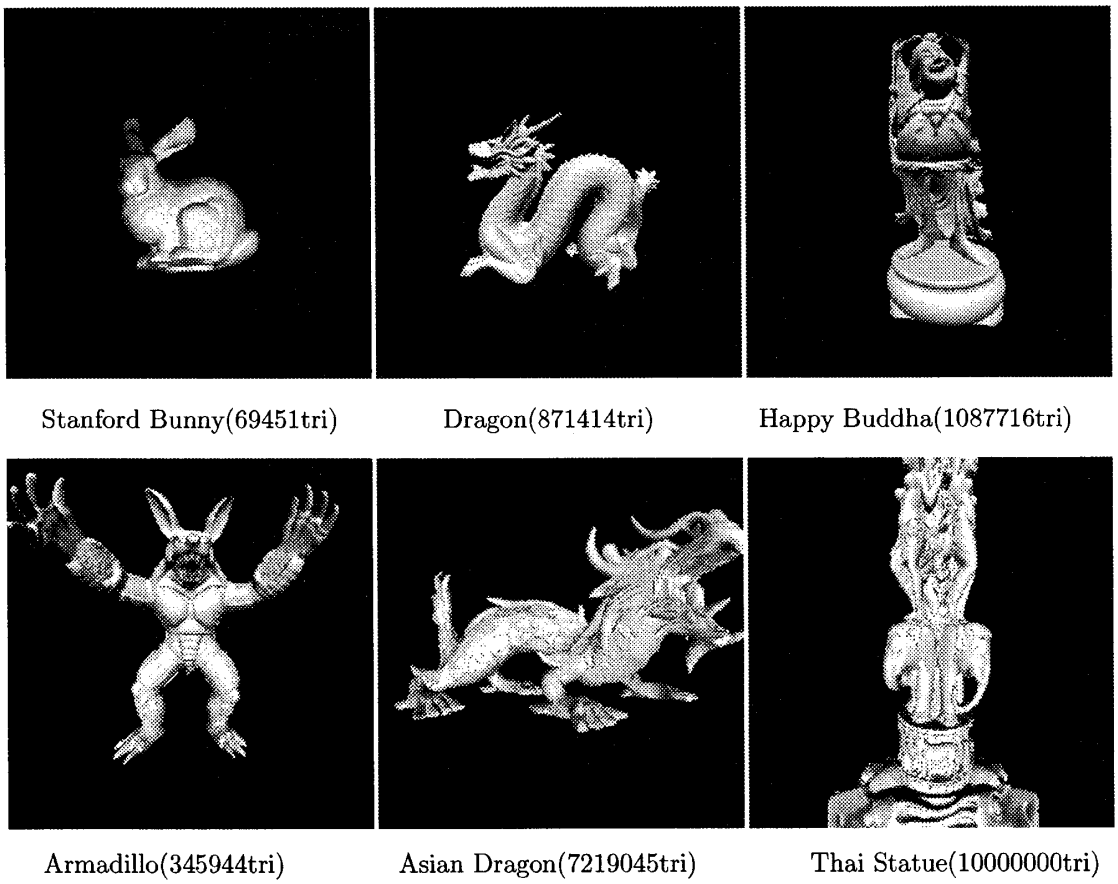
Since the terms including the edges and the normal are independent in Equation (2), these values can be precomputed. Thus, it achieves faster intersection tests than the original one.

In order to boost the intersection tests, it is essential to exploit the property of ray packets with SIMD operations. However, this algorithm is not suitable for dealing with ray packets. By optimizing this algorithm for the packet-triangle intersection tests, the intersection calculation time can be shorter than that of the Pluecker intersection test.

3.2 Proposed Packet-Triangle Intersection Test

We propose a new packet-triangle intersection algorithm based on the Möller intersection. As described in Section 3.1, the algorithm can be further optimized for packet-triangle intersection tests.

Effectively treating ray packets brings a great benefit to achieve high speed intersection tests. Basically, a set of coherent rays, which are like primary rays, shadow rays, and diffuse rays, performs similar intersection tests with triangles, and thus the memory reference regarding the set shows the high locality. Treating



Stanford Bunny(69451tri) Dragon(871414tri) Happy Buddha(1087716tri)

Armadillo(345944tri) Asian Dragon(7219045tri) Thai Statue(10000000tri)

Figure 4: Test scenes

multiple coherent rays together causes better memory accesses and efficient use of SIMD operations and result in boosting the intersection test.

More amount of calculation is able to be precomputed by using the property of the ray packets that all the origins of rays in the ray packet are same. Thus, although the terms in Equation (2) related with the ray direction D cannot be precomputed, the other terms are able to be precomputed because only the ray directions D are not constant during a packet-triangle intersection test.

In Equation (2), $N \cdot S$ is constant because the rays in the packet share the same origin O. By calculating and saving $N \cdot S$ of the ray packet in advance, the amount of calculation can be reduced. In Equation (2), only $N \cdot S$ can be precomputed. However, by applying the scalar triple product rules, we can obtain the improved equation.

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

In this equation, since $N \cdot S$, $-S \times E_2$, and $S \times E_1$ are constant, they can be precomputed for each ray packet. Figure 3 shows the flowchart of our proposed intersection test. The new precomputation process is shown as the dotted box. The precomputation expects to allow for efficient reduction of the computation cost of the intersection tests.

Compared with the Pluecker intersection test using packets, our proposed test method requires less instructions in implementation. Although the Pluecker intersection test requires 16 SIMD operations for one primitive-triangle test, our proposal test requires only 14 SIMD operations at maximum.

Furthermore, our algorithm can introduce the early termination. In the proposed algorithm, several conditional tests are performed in one intersection test as Möller-Trumbore algorithm. If one of the checks does not pass, no more calculation is necessary. This makes the intersection test quick. Early termination of the projection algorithm and the Pluecker algorithm cannot be done effectively because the conditional tests are performed at the end of the intersection test.

4 Experimental Results and Discussion

In order to evaluate the proposed intersection algorithm, we implemented it on a PC consisting of In-

tel Pentium 4 processor running at 3.4GHz with 2GB DDR memory. Experiments were conducted by generating images from Stanford data archives[7] shown in Figure 4. All these images of 256×256 resolutions were generated by 2×2 ray packet of primary rays. In order to evaluate the performance of intersection algorithm, no spatial data structure is used.

Figure 5 shows the comparison of the intersection time. In the figure, the x-axis indicates the test scenes. The y-axis indicates the speedup ratio of the intersection time of the improved projection method, the Pluecker method using ray packets, and our proposed method to those of the improved Möller method[6]. Note that only the Pluecker intersection algorithm and our proposed algorithm save the redundant computations by exploiting the property of the ray packets.

This figure shows that our algorithm achieves faster intersection tests than those of the other algorithms. Although the arrangement of the triangles in the scenes affects the performance of the intersection test, our proposed method achieves about 1.1x to 1.4x speedup for all the test scenes. Compared with the Pluecker packet-triangle intersection test, our algorithm achieves speedup even in Asian Dragon scene although the Pluecker algorithm drops the performance.

One of the reasons of the speedup is due to less operations required for the intersection test. As mentioned in Section 3.2, the number of instructions of our algorithm is less than that of the other ones. The second reason is that early termination reduces the number of operations. Usually several conditional tests should be performed in one iteration. However, if the condition is not satisfied to pass the check, another iteration can begin to be processed. From the analysis of the early termination, more than 75% of the ray-packet intersection tests benefit from the early-termination before the final test. This early termination makes the intersection time short.

Although only primary ray packets are used in these experiments, our proposed method works for the shadow ray packets and diffuse ray packets because of sharing common origin and high coherent of these packets.

5 Conclusions

In this paper, we proposed the new fast intersection algorithm for ray-packet intersection tests. To increase the portion of precomputation by exploiting the feature of ray packets, the redundant computation can be removed for the fast intersection test.

We have implemented and evaluated our proposed intersection algorithm on a PC. The experimental results illustrated the achievement of faster intersection tests in comparison with other intersection algorithms. Our proposed algorithm achieves a 1.4x speedup at the maximum. The algorithm exploits the powerful computation power of current CPU resources like SIMD operations and high bandwidth caches.

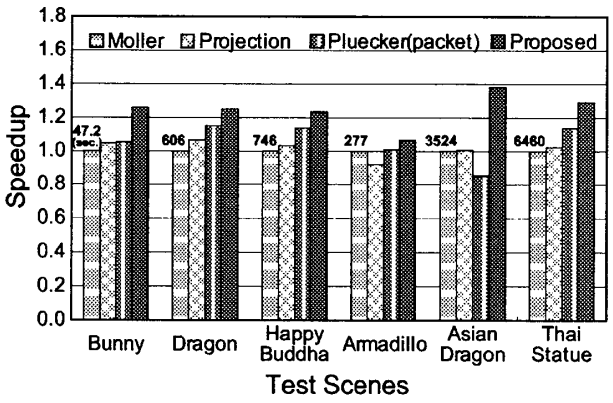


Figure 5: Comparison of intersection algorithms

Our future work includes the combination of the proposed algorithm and the spatial data structure, which is one of the methods to reduce the number of the intersection tests. It is expected that this combination leads to the interactive ray tracing. Moreover, more detailed evaluation for not only the primary ray packets but also the shadow ray packets and the diffuse ray packets is required. In addition, the development of intersection algorithms for the dynamic scenes is our future work.

References

[1] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination.* PhD thesis, Saarland University University, 2004.

[2] Carsten Benthin. *Realtime Ray Tracing on Current CPU Architectures.* PhD thesis, Saarland University University, 2006.

[3] Didier Badouel. An efficient ray polygon intersection. *Graphics Gems*, pages 390–393, 1990.

[4] Jeff Erickson. Pluecker coordinates. Ray Tracing News, 1997. http://www.acm.org/tog/resources/RTNews/html/rtnv10n3.html#art11.

[5] Tomas Möller and Ben Trumbore. Fast, minimum storage ray triangle intersection. *Graphics Tools*, 2(1):21–28, 1997.

[6] Ronen Barzel, editor. *Graphics Tools - the jgt editors' choice.* A K Peters, Ltd, 2005.

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