Efficient triangle–triangle intersection test for OBB-based collision detection

Jung-Woo Chang, Myung-Soo Kim*

School of Computer Science and Engineering, Seoul National University, Seoul 151-744, Republic of Korea

ARTICLE INFO

Abstract

We present an efficient algorithm for triangle–triangle intersection test in oriented bounding box (OBB)-based collision detection. In testing two OBB leaf nodes (i.e., rectangles), many intermediate computation results can be reused for the intersection test of two triangles they contain. It is considerably easier to detect redundant operations when we work in the local coordinate of the bounding rectangle rather than in the global coordinate of the object. The performance improvement of our algorithm is based on this observation that eliminates redundant computations. Compared with conventional algorithms, we have observed 15–79% improvement in computing time. We demonstrate the effectiveness of our approach using several experimental results.

Keywords:
Triangle–triangle intersection
OBB
Collision detection
Coordinate representation

1. Introduction

Collision detection is important for various applications in computer graphics, animation, games, and robotics, to mention only a few. The bounding volume hierarchy (BVH) of oriented bounding box (OBB) is often employed for collision detection of 3D objects with triangular meshes [1]. Other conventional bounding volumes include sphere, axis-aligned bounding box (AABB), k-discrete orientation polytope (k-DOP), and line swept sphere (LSS) [2].

The BVH is a tree constructed with bounding volumes. Each node of BVH is a volume that bounds the corresponding set of primitives. The leaf nodes of BVH generally bound single primitives. In this article, we focus on the OBB-based BVH and collision detection between triangular meshes, where we show how to speed up the triangle intersection test using the result of checking OBB leaf nodes.

Leaf nodes of OBB-trees are rectangles that bound single triangles. The intersection test for triangles is thus preceded by a similar test for their bounding rectangles. Recycling computation results from the rectangle test, we can make the triangle test more efficient than other conventional methods [3–6] that start the triangle intersection test from the scratch. In experimental results, we have observed 15–79% performance improvement in computing time.

The first step for testing the intersection between two triangles is to transform one triangle to the coordinate system of the other triangle. This is an expensive procedure. In OBB-based environments, we show that this step can be simplified by recycling the results from a similar transformation for the bounding rectangles. Moreover, representing each triangle in the local coordinates of the bounding rectangle, we can easily extract common operations and thus eliminate redundant computations in the remaining steps of the triangle intersection test.

2. Related work

Möller [3] computes the signed distances of three vertices of a triangle from the plane containing the other triangle. If the signs are all positive or all negative, there is no intersection. Otherwise, we repeat another triangle–plane test by switching the roles of two triangles. In case the two triangle–plane tests report intersections, the problem is then reduced to an overlap test between two line segments (i.e., triangle–plane intersections) lying on the same line of intersection (between the two planes containing the triangles).

Held [4] first computes the intersection line segment between a triangle and the plane containing the other triangle, and reduces the problem to a 2D intersection test between the line segment and the other triangle.

Guigue and Devillers [5] follow the approach of Möller [3], while they use an orientation predicate which is defined by the determinant of a 4 × 4 matrix. The signed distance of each vertex is computed from this orientation predicate. Moreover, the overlap test for two line segments on the same line of intersection can also be carried out efficiently using the signs of two orientation predicates. The intersection test for two triangles is reduced to that of checking the signs of eight orientation predicates.
Tropp et al. [6] take an algebraic approach, while their basic framework is similar to that of Held [4]. Using a technique based on linear algebra and reusing some computation results for certain variables, Tropp et al. [6] make the algorithm more efficient.

In general configurations of two input triangles, Guigue and Devillers [5] and Tropp et al. [6] usually show better performance than others. Nevertheless, in our case, the two triangles are represented in the local coordinates of the bounding rectangles, where we found that Möller [3] is more suitable to our purpose because of the simplicity of algorithm. In this article, we adopt the algorithm of Möller [3].

3. Algorithm

3.1. Bounding rectangle

The OBB leaf node that bounds a single triangle is a rectangle that shares one edge and two vertices with the triangle [7]. Fig. 1(a) shows that the minimum bounding rectangle shares the longest edge with an obtuse triangle. For an acute triangle, any edge can work for the purpose; but as shown in Fig. 1(b) we take the shortest edge, which corresponds to a minimum rectangle that is closer to a square. Each minimum rectangle has area twice that of the triangle.

A bounding rectangle is represented using a center position vector, a $3 \times 3$ orientation matrix $[r_x, r_y, r_z]$, and two real numbers for side lengths. The direction vector $r_x$ is set to the unit vector along the shared edge with the triangle. The unit vector $r_x$ is normal to the plane containing the triangle; and $r_y = r_x \times r_z$. The main difference of our approach from other methods is that we represent each triangle compactly in the local coordinates of the bounding rectangle using an additional parameter $a$ as shown in Fig. 2, where $-\delta x \leq a \leq \delta x$.

3.2. Input data representation

Conventional algorithms such as Möller [3], Held [4], and Guigue and Devillers [5] require as input the positions of all six vertices of two triangles. On the other hand, Tropp et al. [6] require, for each triangle, the position of one vertex and two directed edges from that vertex to the other vertices. Our algorithm requires the following input data:

- The half lengths of two bounding rectangles: $(dx_1, dy_1)$ and $(dx_2, dy_2)$.
- The relative position and orientation of one rectangle with respect to the other: $\vec{c}$ and $[r_x, r_y, r_z]$.
- For each triangle, an additional parameter for the position of the third vertex: $a_1$ and $a_2$. (This is the only extra storage requirement of our method in the OBB data structure.)
- The inner product of two vectors $\vec{c}$ and $\vec{r}_z$: $d = \vec{c} \cdot \vec{r}_z$. (The value of $d$ has already been computed as a part of the intersection test for two bounding rectangles.)

Using the input data, the six vertices $p_1, p_2, p_3, q_1, q_2, q_3$ of two triangles can be represented as follows:

$p_1 = (-dx_1, -dy_1, 0)$,
$p_2 = (dx_1, -dy_1, 0)$,
$p_3 = (a_1, dy_1, 0)$,
$q_1 = -\vec{r}_x \cdot dx_2 - \vec{r}_y \cdot dy_2 + \vec{c}$,
$q_2 = \vec{r}_x \cdot dx_2 - \vec{r}_y \cdot dy_2 + \vec{c}$,
$q_3 = \vec{r}_x \cdot a_2 + \vec{r}_y \cdot dy_2 + \vec{c}$.

Remark. In the remaining steps of computation, we do not use all the coordinate values of the six vertices. The triangle–plane intersection tests need none of them. In the final stage of testing the intersection between two triangles, we use either $x$ or $y$ coordinate values of the vertices. Thus the required information of six $x$ or $y$ coordinate values can be obtained using one unary subtraction, five additions/subtractions, and three multiplications.

3.3. Triangle–plane intersection

Möller [3] uses the signed distances of vertices from a plane to determine whether a triangle intersects a plane. In our case, the triangle $\triangle p_1p_2p_3$ is contained in the $xy$-plane and the vector $\vec{r}_z$ is normal to the plane containing $\triangle q_1q_2q_3$. Thus the signed distances can easily be computed as follows:

$dp_1 = -dx_1 \cdot r_x \cdot dy_1 \cdot r_y - dy_1 \cdot r_y - d$,  
(1)
$dp_2 = dx_1 \cdot r_x \cdot dy_1 \cdot r_y - dy_1 \cdot r_y - d$,  
(2)
$dp_3 = a_1 \cdot r_x \cdot dy_1 \cdot r_y - dy_1 \cdot r_y - d$,  
(3)
$dq_1 = -r_x \cdot dx_2 - r_y \cdot dy_2 + c_z$,  
(4)
$dq_2 = r_x \cdot dx_2 - r_y \cdot dy_2 + c_z$,  
(5)
$dp_3 = a_2 \cdot r_x \cdot dy_2 + c_z$,  
(6)

where $\vec{r}_z = (r_x, r_y, r_z)$ and $\vec{c} = (c_x, c_y, c_z)$. Some terms such as $dy_1 \cdot r_y$ appear two or three times in these expressions. Thus we can reduce the number of arithmetic operations by computing them only once.

When all three of $dp_1, dp_2, dp_3$ have the same sign, $\triangle p_1p_2p_3$ has no intersection with the plane containing the other triangle $\triangle q_1q_2q_3$. When all three of $dp_1, dp_2, dp_3$ are equal to zero, the two triangles are contained in the same plane. In this singular case, which seldom occurs, we employ the algorithm of Möller [3].
When $\triangle p_1 p_2 p_3$ intersects the plane containing $\triangle q_1 q_2 q_3$, and vice versa, we continue the intersection test for two triangles, which is discussed below.

### 3.4. Triangle–triangle intersection

Let $P$ be the plane containing $\triangle p_1 p_2 p_3$ and $Q$ be the plane containing $\triangle q_1 q_2 q_3$, and let $L = P \cap Q$ be the line of intersection. When $\triangle p_1 p_2 p_3$ intersects the plane $Q$, the triangle $\triangle p_1 p_2 p_3$ also intersects the line $L$ in a line segment. The two triangles intersect if and only if the two line segments of intersection overlap as shown in Fig. 3 [3].

We need to compute the end points of each line segment to check whether the two line segments overlap. The end points are the intersection points of two edges of a triangle with the plane containing the other triangle. Without loss of generality, we may assume $dp_1$ and $dp_2$ are of different signs and the edge $p_1 p_2$ intersects the plane $Q$. A point $v$ on the edge $p_1 p_2$ is parameterized as $p_1 + t(p_2 - p_1)$, $t \in [0, 1]$; and the signed distance of the point $v$ from the plane $Q$ is given as $dp_1 + t(dp_2 - dp_1)$. The point $v$ is contained in the plane $Q$ when the signed distance is zero; thus the intersection point of $p_1 p_2$ with the plane $Q$ is computed as

$$v = p_1 + \frac{dp_1}{dp_1 - dp_2}(p_2 - p_1).$$

(7)

There is no need to compute the exact position of the intersection point [3]. The two line segments are contained in the same line $L$; thus the result of the overlap test is preserved even after the line $L$ is projected onto a proper coordinate axis [3]. In our case, the direction vector of $L$ is determined by the cross product of the z-axis direction (0, 0, 1) and the normal vector $I_z = (r_{z0}, r_{z1}, r_{z2})$. The direction vector $(-r_{z1}, r_{z0}, 0)$ thus implies that the $x$-axis is a proper axis when $|r_{z1}|$ is larger than $|r_{z0}|$, and the $y$-axis is a proper axis otherwise. Consequently, we need to evaluate formula (7) only for the coordinate of a proper axis.

### 3.5. Elimination of divisions

Formula (7) contains one addition, two subtractions, one multiplication, and one division. In contemporary computer systems, division takes 4–8 times more execution time than other simple arithmetic operations such as addition/subtraction, multiplication, and comparison [8]. Thus we remove the divisions as follows. When the four end points are given as $A + B/y_0, A + C/x_1, D + E/y_0$, and $D + F/y_1$, the overlap test produces exactly the same result using $A x_0 y_0 y_1', B x_0 y_0 y_1', A x_0 y_0 y_1' + C x_0 y_0 y_1', D x_0 y_0 y_1' + E x_0 y_0 y_1'$, and $D x_0 y_0 y_1' + F x_0 y_0 y_0'$ instead of the original four end points. More details of this approach can be found in the following URL:


The following discussion on operation counts is based on the revised version of Möller [3] using the division removal as discussed above.

### 4. Operation counts

Before we start operation counts on specific algorithms, we need to understand the difference in coordinate systems employed by the algorithms for the representation of their input triangles. Let $T_1$ and $T_2$ be two triangles contained in two moving objects $O_1$ and $O_2$ under motions $M_1$ and $M_2$, respectively. Moreover, let $R_1$ and $R_2$ be the bounding rectangles of $T_1$ and $T_2$, respectively. In conventional collision detection systems, $T_i$ ($i = 1, 2$) is represented in the coordinate system of the object $O_i$. On the other hand, we represent the triangle $T_i$ ($i = 1, 2$) in the local coordinates of the bounding rectangle $R_i$.

To apply an intersection test to two triangles, we need to represent the two triangles in the same coordinate system. For this purpose, conventional algorithms check the intersection between $T_1$ and $M_{12}(T_2)$ in the coordinate system of the object $O_1$, where $M_{12} = M_1^{-1} M_2$ is the relative motion of $M_2$ with respect to $M_1$. (Note that the motion matrix $M_1$ is constructed only once for each time frame of the two moving objects $O_1$ and $O_2$.) In contrast to that, our algorithm represents the two triangles $T_1$ and $M_{12}(T_2)$ in the local coordinates of the rectangle $R_i$. Note that the OBB test for the two rectangles $R_1$ and $R_2$ is also done in the local coordinates of $R_i$. Thus, in our algorithm the transformation $M_{12}(T_2)$ is carried out as a side product of the OBB test for the two rectangles $R_1$ and $R_2$. In other conventional algorithms, we need to compute $M_{12}(T_2)$ from the scratch, which is considerably more expensive than ours.

In the methods such as Möller [3], Held [4], and Guigue and Devillers [5], the transformation $M_{12}(T_2)$ requires 27 multiplications and 27 additions. Tropp et al. [6] represent each triangle using one vertex and two vectors; thus, 27 multiplications and 21 additions are needed. In our algorithm, the transformation had been completed in the previous step of the OBB test; thus no additional operations are needed.

The first three rows of Table 1 report the results of operation count for previous algorithms, whereas the last row reports the number of operations required for our algorithm. Triangle–plane I reports the operation counts for the first triangle–plane intersection test, including the operations for the coordinate transformation $M_{12}(T_2)$. On the other hand, Triangle–plane II reports additional operation counts for the second triangle–plane intersection test. Note that Tropp et al. [6] use only one triangle–plane test. Thus there is no second test and thus no operation count for this case. (We do not include Held [4] in this comparison since Tropp et al. [6] is more efficient.) Finally, Triangle–triangle reports the operation counts for the final triangle–triangle intersection test. The number of operations for our algorithm in these three steps can be counted as follows:

1. First triangle–plane intersection test:
   - Three signed distances $dp_1$, $dp_2$, and $dp_3$ are computed using three multiplications, one unary subtraction, and five additions/subtractions as shown in Formulas (1)–(3).
   - Two multiplications and two comparisons are also needed for the triangle–plane intersection test, including the detection of the singular case [3].

2. Second triangle–plane intersection test:
   - Three signed distances $dq_1$, $dq_2$, $dq_3$ are computed using five additions/subtractions and three multiplications as shown in Formulas (4)–(6).
   - Two multiplications and two comparisons are also needed for the triangle–plane intersection test, including the detection of the singular case [3].
method of Trenkel et al. [10], which is a benchmark generation.

The different configurations were generated by the sphere with 15,536 triangles, and located the two models in that report intersection.

169,944 triangles. There are 176,137 pairs of bounding rectangles 2528 frames in the interior of an airplane constructed with 404 triangles moves along a path consisting of relative distances:

- When the x-axis is a proper axis, we need five additions/subtractions and three multiplications to compute \( q_{1x}, q_{2x} \), and one unary subtraction for \( p_{1x} \). Similarly, when the y-axis is a proper axis, we need the same number of operations.

- The division-free version of Formula (7) requires 12 additions/subtractions and 16 multiplications to check whether two line segments overlap. There are some cases where we need two additional multiplications to decide which two edges intersect the plane containing the other triangle [3].

5. Experimental results

We have implemented our triangle intersection algorithm in C++ on an Intel Core2 2.4 GHz PC with a 2.0 GB main memory. For comparison purpose, in addition to ours we have also included three other implementations of triangle–triangle intersection test: Möller [3], Guigue and Devillers [5], and Tropp et al. [6]. The implementation details including source codes are available from the following URLs:

- [http://mis.hevra.haifa.ac.il/~ishimshoni/TRI/](http://mis.hevra.haifa.ac.il/~ishimshoni/TRI/)

5.1. Three scenarios

To demonstrate the performance of our algorithm, we tested the different implementations in three heterogeneous scenarios, each applied to different input meshes and position/orientation configurations.

In scenario I, we repeat one test of Klosowski et al. [9], where a hand model with 404 triangles moves along a path consisting of 2528 frames in the interior of an airplane constructed with 169,944 triangles. There are 176,137 pairs of bounding rectangles that report intersection.

In scenario II, we employed two Happy Buddha models, each constructed with 15,536 triangles, and located the two models in 229,824 different configurations of relative position and orientation. The different configurations were generated by the sphere method of Trenkel et al. [10], which is a benchmark generation method for collision detection algorithms. We used four different relative distances: \(-2\%\), \(-1\%\), 0\%, and 1\% of the size of input models. (Two models collide in the cases of \(-2\%\), \(-1\%\), and 0\%, whereas they have no collision in the case of 1\%). Each distance is determined by the radius of a bounding sphere. A total of 266 different positions are generated on the sphere by sampling the spherical coordinates at every 15°. Moreover, a total of 144 different orientations are generated by sampling Euler rotation angles at every 60°. For each of the four distances, a total of 38,304 configurations are tested. Fig. 4 illustrates some sample snapshots of scenarios I and II.

![Fig. 4. Illustration of some snapshots of scenarios I and II: (a) and (b) two sample frames of scenario I, (c) one sample configuration of scenario II, and (d) an enlarged view of the same configuration.](image)

5.2. Performance comparison

Table 2 reports, for each scenario, the number of intersecting pairs remaining after each step of computation. In the first row of scenario I, there are 176,137 intersecting rectangle pairs and 84,931/84,340 intersecting triangle pairs. Also shown are the numbers of intersecting pairs remaining after the first and second triangle–plane intersection tests. The methods such as Möller [3], Guigue and Devillers [5], and ours detect 84,931 triangle pairs as intersecting pairs; on the other hand, Tropp et al. [6] reports 84,340 triangle pairs as intersecting pairs. This difference is due to the use of different numerical tolerances in the implementation of these methods.

Table 3 reports, in each scenario using four different algorithms, the execution time of triangle intersection test, which is

### Table 1
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Triangle–plane I</th>
<th>Triangle–plane II</th>
<th>Triangle–triangle</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Möller</td>
<td>± 2 47 2</td>
<td>± 21 20 2</td>
<td>± 15 23/25 8/18</td>
<td>± 84 90/92 12/22</td>
</tr>
<tr>
<td>Guigue and Devillers</td>
<td>± 51 44 2</td>
<td>± 24 17 2</td>
<td>± 28 18 6/12 103</td>
<td>± 103 79 10/16</td>
</tr>
<tr>
<td>Tropp et al.</td>
<td>± 37 46 8</td>
<td>± ~ ~ ~</td>
<td>± ~ ~ 19/21 37/38</td>
<td>± 56/58 83/84 19</td>
</tr>
<tr>
<td>Ours</td>
<td>± 6 5 2</td>
<td>± 5 5 2</td>
<td>± 18 15/21 2/19</td>
<td>± 29 29/31 13/23</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Scenario I</th>
<th>Triangle–plane I</th>
<th>Triangle–plane II</th>
<th>Triangle–triangle</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>± 48 47 2</td>
<td>± 21 20 2</td>
<td>± 15 23/25 8/18</td>
<td>± 84 90/92 12/22</td>
</tr>
<tr>
<td></td>
<td>± 51 44 2</td>
<td>± 24 17 2</td>
<td>± 28 18 6/12 103</td>
<td>± 103 79 10/16</td>
</tr>
<tr>
<td></td>
<td>± 37 46 8</td>
<td>± ~ ~ ~</td>
<td>± ~ ~ 19/21 37/38</td>
<td>± 56/58 83/84 19</td>
</tr>
<tr>
<td></td>
<td>± 6 5 2</td>
<td>± 5 5 2</td>
<td>± 18 15/21 2/19</td>
<td>± 29 29/31 13/23</td>
</tr>
</tbody>
</table>
applied only after their bounding rectangle pairs report intersection. (This measure includes triangle–plane tests and triangle–triangle tests, but not the tests for the pairs of bounding rectangles.) Figs. 5 and 6 illustrate these execution times as graph. Our method is faster than Möller [3] about 24% for scenario I, about 32–40% for scenario II, and about 73–75% for scenario III. It is also faster than Guigue and Devillers [5] about 21% for scenario I, about 27–39% for scenario II, and about 69–72% for scenario III. Moreover, our method is faster than Tropp et al. [6] about 15% for scenario I, about 28–51% for scenario II, and about 77–79% for scenario III.

In scenario I and in three cases of scenario II, about 40–48% of intersecting rectangle pairs turn out to be actually intersecting triangle pairs; and in other cases, no intersecting rectangle pairs lead to intersecting triangle pairs. In the cases where some triangle pairs intersect, our method is faster than previous

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Rectangle pairs</th>
<th>Triangle–plane I</th>
<th>Triangle–plane II</th>
<th>Triangle pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario I</td>
<td>176,137</td>
<td>168,202</td>
<td>152,140</td>
<td>84,931/84,140</td>
</tr>
<tr>
<td>Scenario II</td>
<td>-2%</td>
<td>7,133,489</td>
<td>6,282,739</td>
<td>5,519,908</td>
</tr>
<tr>
<td>Scenario II</td>
<td>-1%</td>
<td>4,802,570</td>
<td>4,210,840</td>
<td>3,678,720</td>
</tr>
<tr>
<td>Scenario II</td>
<td>0%</td>
<td>2,124,199</td>
<td>1,990,935</td>
<td>1,690,868</td>
</tr>
<tr>
<td>Scenario II</td>
<td>1%</td>
<td>224,955</td>
<td>142,096</td>
<td>68,087</td>
</tr>
<tr>
<td>Scenario III</td>
<td>1 × 10^{-5}</td>
<td>76,566</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Scenario III</td>
<td>2 × 10^{-5}</td>
<td>71,990</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Scenario III</td>
<td>4 × 10^{-5}</td>
<td>67,206</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Scenario III</td>
<td>8 × 10^{-5}</td>
<td>59,862</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Scenario III</td>
<td>16 × 10^{-5}</td>
<td>25,508</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2
Number of intersecting pairs remaining after each step.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Möller</th>
<th>Guigue and Devillers</th>
<th>Tropp et al.</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario I</td>
<td>0.0459694</td>
<td>0.0443563</td>
<td>0.0413829</td>
<td>0.0351119</td>
</tr>
<tr>
<td>Scenario II</td>
<td>2.2003080</td>
<td>2.0586641</td>
<td>2.0925340</td>
<td>1.5039452</td>
</tr>
<tr>
<td>Scenario II</td>
<td>1.4761565</td>
<td>1.3910706</td>
<td>1.4134377</td>
<td>1.0060829</td>
</tr>
<tr>
<td>Scenario II</td>
<td>1.7196929</td>
<td>0.6811928</td>
<td>0.7012427</td>
<td>0.4847263</td>
</tr>
<tr>
<td>Scenario II</td>
<td>0.0708643</td>
<td>0.0699246</td>
<td>0.0864258</td>
<td>0.0424375</td>
</tr>
</tbody>
</table>

Table 3
Execution time (in s).

**Fig. 5.** Execution time for scenario II.

**Fig. 6.** Execution time for scenario III.
methods about 15–33%. On the other hand, in the cases where there is no intersecting triangle pair, our method is faster than others about 39–79%. The better performance in these non-intersecting cases is due to the fact that the number of operations needed for our method is considerably smaller than others when early rejections are made as the result of no triangle–plane intersections. The simple algorithm of Möller [3] is faster than Tropp et al. [6] in the non-intersecting cases. This is because, in the non-intersecting cases, there is no much difference in the number of operations between Möller [3] and Tropp et al. [6].

6. Conclusion

We have presented an efficient algorithm for triangle–triangle intersection test in OBB-based collision detection. Many intermediate results from the rectangle–rectangle overlap test are reused in the triangle–triangle intersection test. The analysis on operation counts and the experimental results show that our approach makes a favorable speed up with respect to conventional algorithms. The performance improvement is the most significant when early rejection(s) are made as the result of no intersection from the triangle–plane test(s). The simple representation of triangle in the local coordinates of its bounding rectangle has a clear contribution to this improvement of computational efficiency.

Acknowledgments

The authors would like to thank anonymous reviewers for their invaluable comments. This research was supported by the Korea Research Foundation under the Grant KRF-2008-313-D00923.

References