一种用于光线与三角形网格求交运算的有效剔除算法*

徐智渊+, 唐泽圣, 唐 龙

(清华大学 计算机科学与技术系,北京 100084)

An Efficient Rejection Test for Ray/Triangle Mesh Intersection

XU Zhi-Yuan+, TANG Ze-Sheng, TANG Long

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

+ Corresponding author: E-mail: xuzy@vis.cs.tsinghua.edu.cn

http://www.tsinghua.edu.cn

Received 2002-09-19; Accepted 2002-11-20

Xu ZY, Tang ZS, Tang L. An efficient rejection test for ray/triangle mesh intersection. *Journal of Software*, 2003,14(10):1787~1795.

http://www.jos.org.cn/1000-9825/14/1787.htm

Abstract: This paper proposes a new rejection test for accelerating ray/triangle mesh intersection. In the approach, a ray is defined as the intersection of two nonparallel planes. For a given ray and a complex scene including dense triangle meshes, this approach can cull most nonintersecting triangles by a simple rejection test that only involves triangle/plane intersection tests. With this approach, exploiting image-space coherences for primary rays in ray tracing is straightforward. In order to exploit object-space coherences, the approach can also be combined with popular spatial partition schemes, e.g. bounding box hierarchies and octrees. Furthermore, this approach can be easily extended to more general polygonal meshes.

Key words: rejection test; triangle mesh; intersection; ray tracing; primary ray

摘 要: 提出一种用于光线与三角形网格求交运算中的有效剔除算法.算法中,一根光线被定义为两个非平行平面的交线.针对由稠密三角形网格组成的复杂场景,算法通过三角形和测试平面的相交判断剔除与投射光线不相交的绝大多数三角面片.利用该算法,光线跟踪中主光线在图像空间的相关性可以方便、直观地被利用.为了利用物体在景物空间的相关性,算法可以结合层次包围盒、八叉树等常见的场景划分方法.而且,该算法可以方便地扩展应用于一般多边形网格.

关键词: 剔除测试;三角形网格;求交运算;光线跟踪;主光线

中图法分类号: TP391 **文献标识码**: A

1 Introduction

Ray tracing is an important algorithm for producing high quality images, but its rendering procedure is very

* **XU Zhi-Yuan** was born in 1975. He is a research assistant at the Tsinghua University. His research interests focus on computer graphics. **TANG Ze-Sheng** was born in 1932. He is a professor and doctoral supervisor at the Tsinghua University. His current research areas are the scientific visualization and the computer graphics. **TANG Long** was born in 1938. He is a professor at the Tsinghua University. His current research areas are the scientific visualization and the computer graphics.

slow. The ray/object intersection computations are the most time-consuming operations in ray tracing. For a complex environment, performing ray/object intersection calculations typically spends more than 95% of the total rendering time[1]. Accelerating ray/object intersection calculations becomes a critical issue.

In computer graphics, because of their mathematical simplicity and practical flexibility, polygonal meshes, especially triangle meshes are the most popular geometry representations of 3D objects, and parametric surfaces or other primitives can also be tessellated to triangle meshes. Therefore, how to efficiently implement the intersection operations between rays and triangle meshes is important.

1.1 Previous work

During decades, there have appeared various types of schemes for accelerating ray/object intersection calculations. Here we just give a brief description of some techniques relating to our approach. A good early survey can be found in[2].

One popular class of these schemes makes use of object space coherences. These algorithms usually organize objects in some types of 3D spatial subdivisions for accelerating intersection tests, typical spatial subdivisions are bounding box hierarchies[3], octrees[4], BSP trees[5], etc. Building these types of spatial structures for a complex scene typically requires large extra memory and considerable pre-process time.

There are also some algorithms, e.g. beams[6], cones[7] and pencils[8] ray tracing, etc., grouping a set of neighboring rays into a generalized ray. These algorithms employ ray coherences well, but a common drawback is that generalized rays always introduce some complicated operations.

For the intersection calculation of a ray and a single triangle, lots of algorithms have been presented. The most popular approach first computes the intersection point of a ray and a triangle's plane, and thereafter the intersection point is tested to determine whether it is inside the triangle actually; usually the barycentric coordinates are used[9]. A number of ways about testing a point inside a triangle are reviewed in Ref.[10]. In Ref.[11] , Mőller and Trumbore present a ray/triangle intersection scheme, which do not require storing the triangle's normal. These approaches treat triangles independently and don't employ the share information among neighboring triangles.

In Ref.[12], two new approaches both employ a fact that two neighboring triangles share an edge. The first approach accomplishes the ray/triangle intersection test by performing the in-out test for each triangle using three edge plane equations. Because each plane is shared between neighboring triangles, the cost of intersection calculations is reduced; the second approach using the plücker coordinates also employs the above fact.

1.2 Overview

In this paper, we present a new ray/triangle mesh intersection test approach based on two following facts: (1) For a dense triangle mesh including lots of small triangles, when a given ray hit this triangle mesh, indeed the ray only intersects with a small number of triangles of the mesh in common situations. If there exists a simple rejection test to discard most of triangles not intersected with the ray, the intersection calculations will be more efficient. (2) A ray can be regarded as the intersection of two nonparallel planes, and the ray/object intersection tests can be transformed to plane/object intersection tests, then intersection tests will speed up significantly. It will be explained detailed in next section. Moreover, in triangle meshes, most vertices are shared by about six triangles or less; our approach exploits this fact to reduce the cost of the intersection calculations furthermore.

The remaining parts of this paper are organized as follows: In Section 2, we give detail explanations of the rejection test. After that, in Section 3, exploiting the primary rays' coherences in image space will be discussed. The test results and comparisons are presented in Section 4. In the end, we make some conclusions and discuss future works.

2 Simple Ray/Triangle Mesh Rejection Test

2.1 Basic idea

A common idea for optimizing intersection tests is to do some simple calculations early on which can determine whether the ray totally misses the object to avoid further computations. Our approach rejects most nonintersecting triangles for a certain ray by simple signed distance calculations between the vertices of triangles and planes.

Traditionally, a ray is regarded as infinitesimally thin and defined by an origin point $\vec{o}$ and a direction vector $\vec{d}$: $r(t)=\vec{o}+t*\vec{d}$. In analytic geometry, a line in 3D space can also be represented as the intersection of two nonparallel planes $\begin{cases}\Pi_1: a_1x+b_1y+c_1z+d_1=0\\ \Pi_2: a_2x+b_2y+c_2z+d_2=0\end{cases}$. This is illustrated in the Fig.1. As a result, we decide the position relation of a ray and a triangle by that of two planes and the triangle, so ray/object intersection tests can transform to plane/object tests.

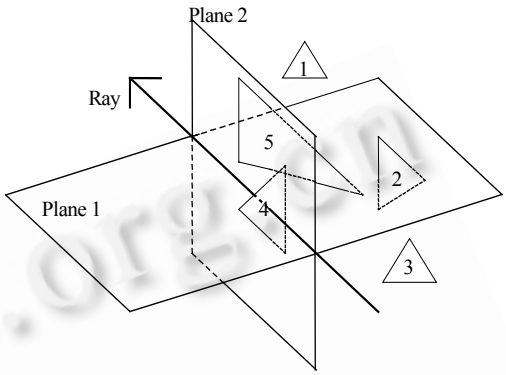

Fig.1 Ray and two planes representation

If a triangle does not intersect with a plane on which a ray lies, we can safely assert that the ray must miss this triangle because the ray is a part of the plane. In Fig.1, triangle 1 and 3 do not intersect with plane 1, thus the ray misses triangle 1 and 3. Triangle 2 intersects with plane 1 but not intersect with plane 2, so it also does not intersect with the ray. In other words, if a ray hits a triangle, this fact will guarantee that every plane that the ray lies on will intersect with this triangle, e.g. triangle 4 in Fig.1. The ray hit triangle 4, thereby triangle 4 must intersect with plane 1 and 2.

According to the above discussion, we present a new rejection test for ray/triangle intersection; outline the rejection test as follows:

Step 1. For a given ray, select two nonparallel planes π_1,π_2 that the ray lies on.

Step 2. Determine triangles in the mesh whether intersect with two test planes. If a triangle does not intersect with either of two planes, it will be rejected; else this triangle is accepted and selected into a candidates list for next intersection tests.

Step 3. After step 2, for a given ray, we have got a triangle candidates list for intersection tests. By exploiting the existing intersection algorithms for single triangle and a ray[9,11], we determine whether a triangle in the candidates list is hit by the ray one by one, and at last get the nearest intersection point.

In Step 2, it may occur that some triangles intersect with two test planes but not hit by the ray, e.g. triangle 5 in Fig.1. Therefore after the rejection test in Step 2, we just get a conservative triangle candidates list for next intersection tests and the next tests in Step 3 are necessary.

2.1.1 Test planes selection

In the above discussion, Step 1 is to select two test planes that the ray lies on. Because there are no extra constrains on the test planes, we can optimize the approach furthermore by selecting appropriate test planes.

Considering a ray: $r(t)=\vec{o}+t*\vec{d}$, there two possibilities: (1) The world coordinate origin lies on this ray, then every plane on which this ray lies must pass through the origin; (2) The origin does not lies on this ray, then the

origin and this ray can define a plane, obviously this plane pass through the origin and the ray. Consequently we can always select a test plane pass through the origin and the ray. A plane π_1 passing through the origin has a simplified equation in the following form: $\pi_1\ a_1x+b_1y+c_1z=0$. The second test plane π_2 can choose a plane perpendicular to the first test plane π_1, and the normal vector of plane π_2 equals the cross product of the ray direction vector and the normal vector of plane π_1.

2.1.2 Triangles and planes relation

Now, we explain how to carry out the rejection test in Step 2. A plane in 3D space splits the space into two parts: positive and negative half-spaces. The triangle/plane position relation has three possibilities: (1) intersecting each other, (2) the triangle lying in the positive half-space, (3) the triangle lying in negative half-space. In the case 1, two vertices of this triangle must lie in different half-spaces of the plane or at least one of the vertices lies on the plane; on the other hand, in the case 2 or 3, all three vertices of this triangle must lie in the same half-space of the plane. Therefore, the triangle/plane position relation is converted to the position relation of three vertices of the triangle and this plane.

The signed distance between a point and a plane is represented as: $Dis=\dfrac{ax+by+cz+d}{\sqrt{a^2+b^2+c^2}}$. The sign of the signed distance indicates the point/plane position relation: $Dis>0$, the point lying in the positive half-space; $Dis<0$, lying in the negative space; $Dis=0$, the point lying on the plane. What we want to know is just which side of the plane the point lies on, so the accurate calculation of the signed distance is not necessary. In fact, we ignore the positive denominator $\sqrt{a^2+b^2+c^2}$ in the expression and just calculate the simplified expression: $ax+by+cz+d$. The sign of this simplified expression have determined the position relation of points against planes.

In Section 2.1.1, the first test plane π_1 pass through the origin and the plane equation was represented as: π_1 $a_1x+b_1y+c_1z=0$. We select the maximal absolute values of three coefficients and three coefficients (a_1,b_1,c_1) of the plane equation divide by it:

$$(a_1,b_1,c_1)\rightarrow\frac{(a_1,b_1,c_1)}{\max\left(|a_1|,|b_1|,|c_1|\right)}.$$

The benefit of this transform is that a coefficient become 1 or –1, it will reduce a multiplication for every point/plane position relation test. Up to now, the position relation test of a vertex and the first test plane only need 2 multiplications, 3 additions or subtractions. Since a triangle has three vertices, every triangle/(the first plane) intersection test only needs 6 multiplications, 9 additions or subtractions, and 3 comparisons. In next Section 2.1.3, we can reduce the cost of triangle/plane intersection tests even more by exploiting a fact that vertices are shared among neighboring triangles.

For the second plane not passing through the origin, similar to the first plane, we can also transform its plane equation coefficients for optimization reasons. As a result, every triangle/(the second plane) intersection test needs 6 multiplications, 12 additions or subtractions, and 3 comparisons.

2.1.3 Efficiency analysis

If the above rejection test is efficient, it must satisfy: first, the calculations involved in the rejection test must be simple enough; it will be explained as follows; second, after the simple rejection test, the number of triangles in the candidates list is small enough, it can be demonstrated by the statistics result in Section 4.

According to practical experience, the first plane's rejection test can discard a substantial number of nonintersecting triangles. Although it is sure that the number of the remainder triangles after the first rejection test seriously rely on the triangle's geometric distribution, a common surface model including n small triangles, as most of triangles in the mesh distribute on the surface, e.g. teapot, bunny model, etc., after the first rejection test, the

average number of reminder triangles is about $o(\sqrt{n})$; this fact has been somewhat demonstrated by the statistics in Section 4. If the number of reminder triangles is small enough, we can ignore the second plane rejection test; else the second test is necessary.

As Euler formula states, for a closed mesh without any holes, there exists a relation between the number of vertices *V*, the number of edges *E*, and the number of faces *F*:

$$V+F-E=2.$$

Assuming every edge is shared by two triangles and a triangle has three edges, we get 3**F*=2**E*. Substitute E by F and ignore the constant 2, we get a result:

$$V\approx F/2.$$

The above expression also infers that a vertex is shared by about six triangles in a closed mesh. We have known in Section 2.12 that every triangle/(the first plane) intersection test needs 6 multiplications, 9 additions or subtractions, and 3 comparisons; but now, in a closed mesh, by using the shared vertices, on the average, every triangle/(the first plane) intersection test only requires 1 multiplication, 1.5 additions or subtractions, and 3 comparisons. Thus, the cost of first rejection test is much lower than that of the triangle/ray intersection test[11]. Certainly, if the mesh is not close, e.g. the triangles in a leaf node of octrees, the number of triangles shared by a vertex will go down, obviously the cost of intersection test will increase too.

After the first plane rejection test, because the vertices of the remainder triangles in the candidates list are not shared broadly by these remainder triangles; vertices' sharing almost do no contributions to the optimization of the second plane's rejection test. Therefore, for the second test plane, every triangle/(the second plane) intersection test still needs 6 multiplications, 12 additions or subtractions, and 3 comparisons.

The test results in Section 4 show that adding the rejection test into the intersection algorithms[9,11] for single triangles and a ray will accelerate the intersection calculations significantly.

2.2 Clustered intersection test

For a complex scene, exploiting triangles' coherences in object space will refine our approach furthermore.

Given a triangles set ***T*** including n neighboring small triangles: ***T***=(t_1,t_2,…,t_n) and a test plane π: $ax+by+cz+d=0$, one naive method determining whether these triangles intersect with the test plane is to test all vertices of n triangles against the plane one by one. When all triangles in the set are located on the same side of the test plane, lots of intersection calculations will be wasted. In fact, we can first test all triangles against the test plane as a whole. If the axis-aligned bounding box (AABB) of the triangles set exists, only two points need to be tested[13]. When the AABB of the triangles set lies completely on one side of the test plane, we can stop the next tests immediately.

For the AABB/plane intersection test, we first find out which of the box diagonals is most closely aligned with the plane's normal; after that the two vertices of this box diagonal are inserted into the plane equation π: $ax+by+cz+d$. If both a positive and a negative result (or a zero) are obtained, the box intersects with the plane.

For the OBB/plane intersection tests, first of all, the test plane normal needs transform to the coordinate system of the OBB; and the rest work is same as the AABB/plane test.

2.3 Combining with spatial subdivision schemes

In Section 2.2, we have discussed how to use the AABBs or OBBs of triangles sets to make clustered intersection tests. For a scene with bounding volume hierarchies or other spatial partitions, the AABBs or OBBs of objects exist; exploiting these data structures is straightforward. Later, in Section 3 the clustered intersection test is integrated into the recursive quad-tree partition on the projective plane for speeding up primary rays. Anyway, the

clustered intersection test can be regarded as a kind of combination of our approach and spatial subdivision schemes.

There still another more direct alternative of combining our scheme with spatial subdivision schemes. In practice, because of memory limits, especially for a highly complex scene including millions of triangles, hundreds of small triangles maybe still exist in the leaf nodes of the bounding volume trees, octrees, or BSP trees, etc. after final spatial partition. In ray tracing, if a given ray intersects with this kind of leaf node, all of these triangles in the leaf node will do intersection tests with the ray one by one, obviously it is inefficient and will cost considerable time. Under this condition, our rejection test scheme is an appropriate alternative. Our approach will discard considerable nonintersecting triangles by simple rejection tests and accelerate the intersection calculations significantly.

2.4 Extending to polygonal meshes

The intersection tests for rays and polygons are also important in many situations. Usually, a polygon consists of n vertices, and vertices are represented as a vertex list $\{v_0,v_1,\ldots,v_{n-1}\}$. The only difference of polygons and triangles is the number of their vertices, but it is no essential impacts on the rejection test in our approach.

Similar to Section 2.2, we can also pre-compute and store the AABB or OBB of the polygon; and then only two points need to be tested against the plane in the rejection test.

3 Exploiting Coherence for Primary Rays

Exploiting kinds of coherences is a key technique for speeding up ray tracing. In Section 2.3, the clustered intersection test can be regarded as a technique exploiting object space coherences. In our approach, exploiting image space coherences for primary rays is also straightforward.

In Fig.2, primary rays leave the viewpoint and pass through the projective plane. All pixels in a horizontal scan-line and the viewpoint decide a plane, and this plane can be shared by these pixels as the first test plane in the rejection test; in other words, pixels in a horizontal scan-line share a same triangle candidates list after the first plane test. It is the same situation for pixels in a vertical scan-line.

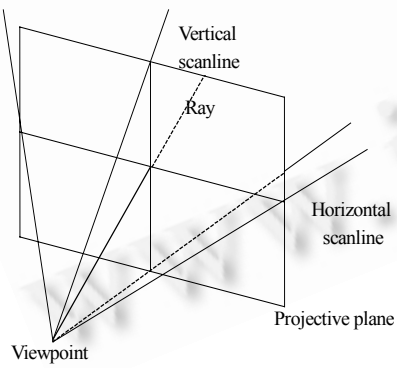

Fig.2 Image space coherence for primary rays

As illustrated in Fig.2, two planes determined by the viewpoint and scan-lines divide the projective plane into four regions, and the intersection of two planes coincides with a ray passing through a pixel's sample point. After making a rejection test by using two test planes in Fig.2, the ray get the triangles candidates list for next intersection tests. In the same time, the triangles in the scene are also divided into four groups corresponding to four regions divided by two planes. It is obvious that a ray passing through a region must miss all triangles in three groups corresponding to the other three regions. This fact will significantly reduce the number of triangles in the rejection test for a certain ray.

Do this procedure recursively and construct a quadtree partition on the projective plane at last. If the number of triangles in groups is small enough or the region includes only one ray, the recursive procedure ends.

The quadtree recursive partition procedure is shown in the pseudo-code as follows.

```
void QuadDiv (int Hs, int He, int Vs, int Ve, Triangle* pTri)
 int Hs,He;      //  Started, Ended horizontal scan-line index
```

int *Vs*,*Ve*; // Started, Ended vertical scan-line index
Triangle* pTri; // The triangle candidates list
{
1: Get the plane equation $\pi_{1,}$ which is determined by the viewpoint and horizontal scan-line indexed [(Hs+He)/2];
2: Get the plane equation $\pi_{2,}$ which is determined by the viewpoint and vertical scan-line indexed [(Vs+Ve)/2];
3: Use two planes $\pi_{1,}$ π_2 determine the intersection triangle candidates list for pixel ([(Hs+He)/2], [(Vs+Ve)/2]), and sort all triangles in the scene into four groups corresponding to four regions divided by planes: $\pi_{1,}$ $\pi_{2.}$
Four pointers pointing to triangles list:
pTlu (Left-Up Region), pTlb (Left-Bottom Region)
pTru (Right-Up Region), pTrb (Right-Bottom Region)
4: QuadDiv (*Hs*,[(*Hs*+*He*)/2]−1, Vs, [(*Vs*+*Ve*)/2]−1,pTlu);
QuadDiv (*Hs*,[(*Hs*+*He*)/2]−1, [(*Vs*+*Ve*)/2]+1,*Ve*,pTlb);
QuadDiv ([(*Hs*+*He*)/2]+1,*He*, *Vs*, [(*Vs*+*Ve*)/2]−1,pTru);
QuadDiv ([(*Hs*+*He*)/2]+1,*He*, [(*Vs*+*Ve*)/2]+1,*Ve*,pTrb);
}

If the scene has built a bounding volume hierarchy or octree structure, etc. in advance, the clustered intersection test in Section 2.2 can be integrated into the recursive procedure. Thus, object space and image space coherences can all be exploited in our approach.

4 Implementation and Results

In this section, we compare the running time under various implementations and present the analyses. We implemented the algorithms with Visual C++6.0. All results were tested on a PC system with 256M RAM and a 1400M AMD Athlon CPU. We selected four test models: teapot, bunny, teapot array1, and teapot array 2. Figures 3~6 are the rendering results of test models.

The results in Table 1 demonstrated the rejection test's efficiency. As a common surface model including n small triangles, after the first plane' rejection test, the average number of remainder triangles is about $o\left(\sqrt{n}\right)$. The number of triangles after the second plane's rejection test is only about twice of the number of triangles intersected with the ray at last. Below is a summary of the results:

Table 1 Rejection test's efficiency

Models	Triangle number	Triangles after the first plane's rejection test (1)	Triangles after the second plane's test (2)	Triangles intersecting with the ray (3)
Teapot	9 216	177.085	6.830	2.981
Bunny	69 451	367.168	5.594	2.557

(1) The average number of triangles per ray after the first plane's rejection test. (2) The average number of triangles per ray after two planes' rejection tests. (3) The average number of triangles per ray intersecting with the ray indeed.

In Table 2, we compared our approach with a common intersection algorithm[11] for single triangle and a ray. In fact, the algorithm[11] was implemented with exhaustive ray/triangle intersection. For the comparison purpose, here our approach didn't utilize ray coherences in image space and only added rejection tests described in Section 2 before exhaustive ray/triangle intersection. The result demonstrated the cost of rejection test is much lower than that of the ray/intersection test, and it resulted in a significant speedup.

Table 2 Comparison between an intersection algorithm and our scheme

Models	Triangle number	Our scheme (1)	Reference algorithm (2)	Speedup ratio
Teapot	9 216	0.0274	0.121	4.416
Bunny	69 451	0.272	0.994	3.654

(1) (2) Running time (in milliseconds/per ray)

In Table 3, in order to evaluate the speedup of our scheme for primary rays, we compared our scheme with a famous freely available ray tracer: POVRAY[14], which is based on the bounding box hierarchy technique.

The results in Table 3 only evaluated the performance for primary rays and ignored shadow, reflection, refraction effects, and anti-aliasing. There was only one light source in the test scenes. POVRAY version is 3.10g, main execute program was compiled with Visual C++ 6.0. Our approach did not employ extra data structures, e.g. bounding boxes for objects; that is to say, there only image-space coherences for primary rays were exploited.

Table 3 Employing image space coherences for primary rays

Models	Triangle number	PovRay (1)	Our approach (2)	Speedup ratio (4)
Teapot	9 216	3.567	0.547	6.521
Bunny	69 451	4.236	0.657	6.447
Teapot array 1	921 600	6.272	3.354	1.870
Teapot array 2	1 843 200	34.561 (3)	6.287	5.497

(1) (2) Rendering time at 512*512 resolution (in seconds/per frame). (3) The sharp increase of rendering time is owing to main memory limitation and frequent data swap between RAM and hard disk. (4) Speedup ratio results show our approach is efficient.

It is obvious that our quadtree recursive partition procedure for utilization of ray coherences in image space is view-dependent, so the rendering time in Table 3 included the cost of the recursive partition procedure.

Of course, different data sets and ray tracing implementations maybe produce somewhat different speedup ratio, but the above statistics can demonstrate our approach is efficient.

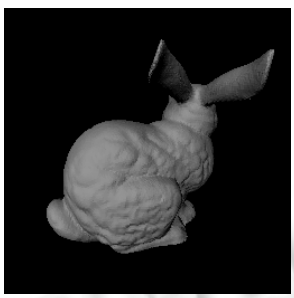

Fig.3 Bunny with 69 451 triangles

Fig.4 Teapot with 9 251 triangles

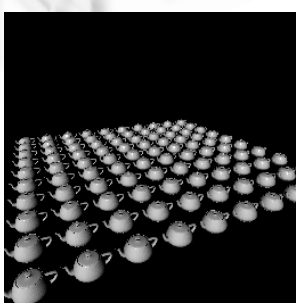

Fig.5 Teapot array 1 with 921 600 triangles

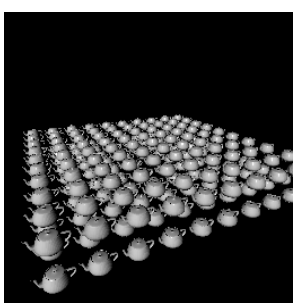

Fig.6 Teapot array 2 with 1 843 200 triangles

5 Conclusions and Future Work

We have presented a new simple rejection test for accelerating the intersection calculations of dense triangle meshes and rays. The scheme can easily exploit ray coherences in image space to speed up primary rays' intersection tests. This approach can also be combined with most popular spatial partition schemes, e.g. bounding box hierarchies and octrees. The efficiency of the approach has been revealed by test results.

One of future refinements about the approach is to introduce parallel mechanism into algorithm's implementation. INTEL and AMD microprocessors both offer SIMD (single instruction multiple data) extension, e.g. SSE, 3DNow. It would greatly benefit the signed point/plane distance calculations that are the most frequent operations in the approach. Ingo Wald etc. [15] state that implementing traversal, intersection, and shading with SSE instructions brings an overall speedup of 2 to 2.5 as compared to a highly optimized C implementation.

References:

[1] Whitted T. An improved illumination model for shaded display. Communications of the ACM, 1980,23(6):343~349.

[2] Arvo J, Kirk D. A survey of ray tracing acceleration techniques. In: Glassner AS, ed. An Introduction to Ray Tracing. San Diego: Academic Press, 1990. 201~262.

[3] Rubin S, Whitted T. A three-dimensional representation for fast rendering of complex scenes. Computer Graphics, 1980,14(3): 110~116.

[4] Glassner AS. Space subdivision for fast ray tracing. IEEE Computer Graphics and Applications, 1984,4(10):15~22.

[5] Sung K, Shirley P. Ray tracing with the BSP tree. In: Kirk D, ed. Graphics Gems III. San Diego: Academic Press, 1992. 271~274.

[6] Heckbert PS, Hanrahan P. Beam tracing polygonal objects. Computer Graphics, 1984,18(3):119~127.

[7] Amanatides J. Ray tracing with cones. Computer Graphics, 1984,18(3):129~135.

[8] Shinya M, Takahashi T, Naito S. Principles and applications of pencil tracing. Computer Graphics, 1987,21(4):45~54.

[9] Badouel D. An efficient ray-polygon intersection. In: Glassner AS, ed. Graphics Gems. San Diego: Academic Press, 1990. 390~393.

[10] Haines E. Point in polygon strategies. In: Heckbert PS, ed. Graphics Gems IV. San Diego: Academic Press, 1994. 24~46.

[11] Mőller T, Trumbore B. Fast, minimum storage ray/triangle intersection. Journal of Graphics Tools, 1997,2(1):21~28.

[12] Amanatides J, Choi K. Ray tracing triangular meshes. In: University of British Columbia, ed. Proceedings of the 8th Western Computer Graphics Symposium. Whistler, BC, Canada, 1997. 43~52.

[13] Kenneth E. A faster overlap test for a plane and a bounding box. 1996. http://www.cs.unc.edu/~hoff/research/vfculler/ boxplane. html.

[14] Persistence of Vision Development Team, Pov-Ray homepage. http://www.povray.org.

[15] Wald I, Slusallek P. State-of-the-Art in interactive ray-tracing. In: Chalmers A, Rhyne MT, eds. State of the Art Reports, Eurographics Conference Proceedings 2001. 2001. 21~42.