An Improved Algorithm for Triangle to Triangle Intersection Test

Xiufen Ye, Le Huang, Lin Wang, Huiming Xing

College of Automation
Harbin Engineering University
Harbin,Heilongjiang Province, China
yexiufen@hrbeu.edu.cn, hlcaca@126.com

***Abstract* - The triangle-to-triangle intersection test is a basic component of all collision detection data structures and algorithms. There are many algorithms that have been put forward in this area until now. Among these algorithms, there is one proposed by YU Hai-yan based on projection theory which can simplify the three-dimensional test into the two-dimensional test and certificate its robustness theoretically. However, the long calculation time can not satisfy the real-time performance. In this paper, an improved algorithm is proposed firstly. Then, the GPU parallel computing is adopted to accelerate our algorithm. Finally, some experiments were carried to validate the improved algorithm. Through the experiments, it shows that the improved algorithm is up to 13 times faster than the original algorithm on average when the triangles' number increases to 2 million and the acceleration times will also increase when much more triangles are detected at the same time and however the algorithm is also stable.**

Index Terms* - *triangle to triangle intersection, collision detection, GPU parallel computing, projection theory.

I. INTRODUCTION

Collision detection is one of the key issues in computer game, cloth simulation, robotics, virtual reality, engineering simulation and so on [1]. In computer game, with the help of collision detection, the real world can be simulated, such as preventing virtual human through the wall or fall down from the floor. In robot simulation program, collision detection is used for path planning to help robot avoid obstacles. The efficiency of collision detection is essential for its applications. Generally, there are two ways to improve the efficiency of collision detection. One is to reduce the number of primitives involved in the testing, the other is to reduce the time of intersection test between two primitives [2]. Triangle is the primitive most commonly used for constructing objects, therefore, the research in quick triangles intersection test algorithm is of great significance.

The algorithms have been put forward for triangle to triangle intersection test can be grouped into two categories: scalar discriminant method and vector discriminant method [2].

The algorithms given by Möller[3], Held[4] and Tropp[5] are the typical ones of scalar discriminant methods. The algorithms proposed by Möller and Held solve intersection test from geometrical view and Tropp's algorithm solves the intersection test by solving linear equations from the view of algebraic.

The algorithms given by Devillers&Guigue[6][7] and Shen[8] are the typical ones of vector discriminant methods. Devillers& Guigue's algorithm is based on the geometric meaning of determinant which consists of vertices of triangles. Shen's algorithm is a method based on separating planes.

Generally, scalar discriminant method can obtain the intersection point coordinates directly, but the existence of cumulative calculation errors has some influence on their accuracy, vector discriminant method is better than scalar discriminant method in stability, because it depends less on calculation accuracy, but this kind of algorithm often requires additional treatment for getting the intersection details. However, all the above algorithms have the following shortcomings. In the aspect of stability, they used the algebraic description way which is difficult to clarify the processing of all kinds of singular states theoretically. In the aspect of algorithm evaluation, they focus on the speed test of the algorithm, but they could not measure and analyze their stability. In practical application, a singular geometry processing mistake can often cause the system to crash.

According to the problem of the above algorithms on stability, Yu Hai-yan [9] put forward the algorithm based on projection theory. It focuses on the singular situation of two three-dimensional triangles' position and sums up the geometric singular phenomenon to several simple cases on a plane. As a result, it solves the geometric singular situation's influence on the stability of the algorithm theoretically.

However, the long calculation time can not satisfy the real-time performance. In this paper, we prompt its speed by optimizing its projections classification, singular cases processing and exclusion separation, and by adopting GPU parallel computing.

This paper is organized as follows: firstly, we will introduce the improved algorithm and then use GPU to accelerate our algorithm, and in the last section, verify the improved algorithms through experiments.

II. ALGORITHM PROCEDURES

As shown in Fig.1, there are two triangles A and B. The symbols xyz and x*y*z* represent the world coordinates system and the calculating coordinates system. We mark x*-y*coordinate plane as H surface, x*-z* coordinate plane as W surface and y*- z* coordinate plane as V surface. The symbols P_0, P_1, P_2 , Q_0, Q_1, Q_2 are the vertices of A and B; Q_{0v}, Q_{1v}, Q_{2v} ,Q_{0w}, Q_{1w}, Q_{2w} are the projections in V,W surface of B's vertices; A_V,A_H,A_W,B_V,B_H,B_W are the projections in V, H, W

surfaces of A and B; L_B is the intersection line between triangle B and plane H; Q_L, Q_R are the endpoints of L_B; Q_{Lv}, Q_{Rv}, Q_{Lw} and Q_{Rw} are the projections in V, W surface of Q_L, Q_R.

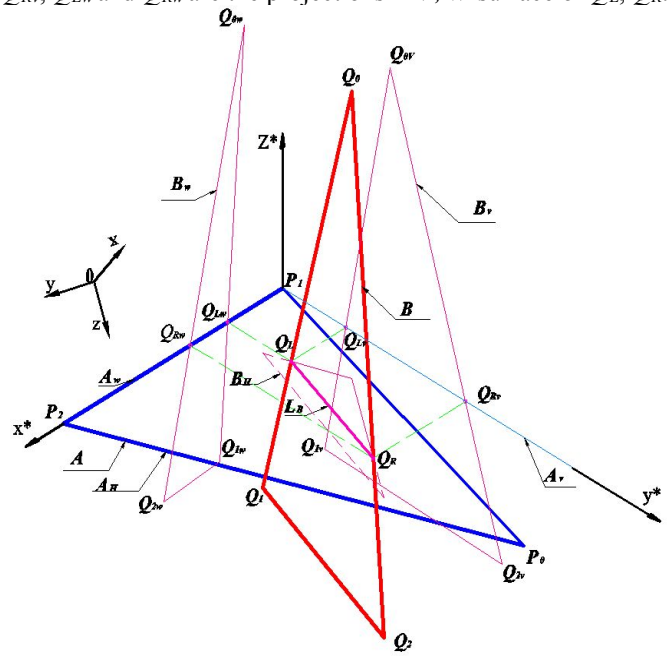

Fig. 1 Two triangles in the calculation coordinates system.

The original algorithm is based on Möller's alogrithm and the projection theory. It can be divided into three steps. Firstly, to establish an appropriate coordinates system and transform the coordinates of A and B from the world coordinates system to the calculating coordinates system. Secondly, to calculate the intersecting line L_B between triangle B and plane H and judge whether there is a part of L_B lies in A_H. Thirdly, to judge whether A intersects with B according the result of the second step. If A intersects with B, the coordinates of intersecting line need to be transformed from the calculation coordinate system to the world coordinate system.

In the following we will compare our improved algorithm with the original one. As we all know, if two triangles intersect, then their projections in each coordinates plane(i.e., x-y plane, x-z plane or y-z plane) must intersect. If there is no intersection between the projections in any of the coordinates planes, the two three-dimension triangles will not intersect either.

A. Establishment of an Appropriate Coordinates System

An appropriate coordinates system can simplify the geometric expression and algebraic operation so we will first establish the coordinates system [9]. This step is similar to the original algorithm.

1) Construction of the Calculating Coordinate System: As shown in Fig.1, we set the plane triangle A lies in as the x*-y* plane and its normal vector as z* axis and one side of A(i.e., vector $\boldsymbol{P_1P_2}$ in Fig.1) as the x* axis. Here are detailed steps: First, setting the unit vector of $\boldsymbol{P_1P_2}$ as $\boldsymbol{n_1}(a_1\ b_1\ c_1)$; Second, calculating the unit normal vector of the triangle A as $\boldsymbol{n_3}(a_3\ b_3\ c_3)$ by use of its vetices P_0, P_1 and P_2; Third, calculating the unit vector $\boldsymbol{n_2}(a_2\ b_2\ c_2)$, which is equal to $\boldsymbol{n_3}$ x $\boldsymbol{n_1}$(cross product). Now $\boldsymbol{n_1}$, $\boldsymbol{n_2}$ and $\boldsymbol{n}3$ which are unit vectors and perpendicular to each other construct the calculating coordinates system x*y*z*. For the calculating coordinates system x*y*z*, P_1 is its origin, x*-axis is the axis $\boldsymbol{n_1}$ lies in, y*-axis is the axis $\boldsymbol{n_2}$ lies in, z*-axis is the axis $\boldsymbol{n_3}$ lies in.

$$\begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} & \boldsymbol{n_3} \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} \tag{1}$$

2*) Coordinates Conversion*: According to P_1 is the origin of x*y*z*, we can get the translation matrix between coordinates system xyz and coordinates system x*y*z* as shown in (2).

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -P_{1x} & -P_{1y} & -P_{1z} & 1 \end{pmatrix} \tag{2}$$

And the rotation matrix is

$$\boldsymbol{R} = \begin{pmatrix} a_1 & a_2 & a_3 & 0 \\ b_1 & b_2 & b_3 & 0 \\ c_1 & c_2 & c_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

Then the coordinates transformation equations are

$$(x^*, y^*, z^*, 1) = (x, y, z, 1) \cdot \boldsymbol{T} \cdot \boldsymbol{R} \tag{4}$$

$$(x, y, z, 1) = (x^*, y^*, z^*, 1) \cdot (\boldsymbol{T} \cdot \boldsymbol{R})^{-1} \tag{5}$$

B. Analyssis of Position Relationship Between Two Triangles According to Projection Theory

This step is our core step and different from the original algorithm[9]. The original algorithm only uses projections on V and H plane. It can only exclude the separating situation(A separates from B) that B_V has no intersecting point with A_V. The situation that B_V has intersecting point with A_V, but B_W has no intersecting point with A_W (obviously, triangle A separates from B under this situation), can not be excluded by the original algorithm. The original algorithm has to judge triangle A separates from B by use of the second step of the original algorithm. Obviously the second step of the original algorithm is complicated and costly. When the situation is not as same as shown in Fig.2(a)(b), the original algorithm has to calculate the intersecting points between all edges(the number is 3) of B_V and the axis that A_V lies in. And then it has to do different treatment for the situation with different number of intersecting points. All these work have reduced the efficiency of the algorithm. So we improve the original algorithm according to solve the above problems. The improved algorithm uses all the projections on V, W and H plane. It can exclude more separating situations than the original algorithm by use of projections on V and W at the same time. We also classify the projections in much more details and predict which edge of B_V would intersect with the axis that A_V lies in,

so that we do not need to calculate the intersecting points and solve the singular situation like the original algorithm. The details of our improved algorithm are shown in the following.

Since the calculating coordinates system is built based on triangle A, we can get the following conclusions: $A_H = A$. A_V and A_W will be a line segment on y*-axis and x*-axis respectively. Even though the shape of projections of triangle B in V, H and W surfaces is undefined, the probable shape is either a triangle or a line segment. The line segment case is just the triangle's special case, it can be processed as same as the triangle case when some parameters are solved in the following.

Taking the projections of V surface as an example (the projections of W surface are similar), all possible projection shapes and positional relationship between the projections are shown in Fig.2.

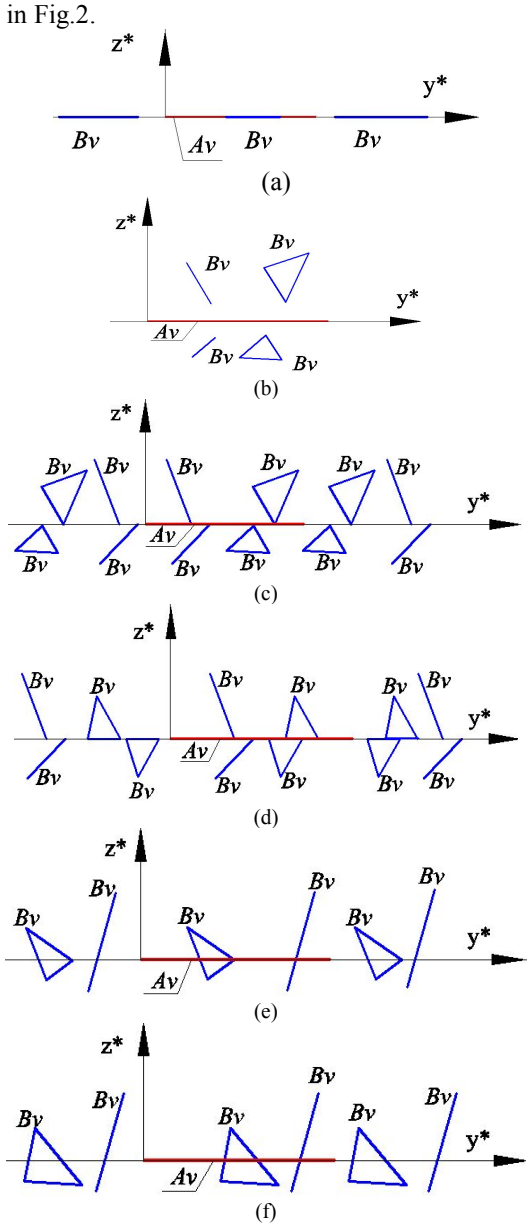

Fig.2 Projections of triangles A and B on V surface.
(The red line is the projection of triangle A and the blue line or blue triangle is the projection of triangle B)

Thus, different approaches to different projection classifications are illustrated as follows.

1) As is shown in Fig.2(a), B_V is a line segment on y*-axis. The case will emerge when all vertices' z* coordinates of triangle B are zero. If A_V and B_V have no overlap part, triangle A separates from triangle B and the test process ends up. Otherwise, we need to judge whether A_W and B_W have overlap part, if they have no overlap part, triangle A separates from triangle B. Otherwise, we need to have a test, which is a two-dimensional test, whether triangle A_H intersects with B_H , if so, triangle A intersects with triangle B, if not, triangle A separates from triangle B. The methods to judge whether two collinear line segments have overlap part and whether two triangles intersect on a two-dimensional plane are shown in Part III.

2) As is shown in Fig.2(b), B_V is a line segment or a triangle beyond or under y*-axis. This case will emerge when z* coordinates of all triangle B's vertices are greater or lower than zero. Obviously, triangle A separates from triangle B.

3) As is shown in Fig.2(c), B_V has an intersection point with y*-axis. This case will emerge when among the z* coordinates of all triangle B's vertices, one is zero and the other two are greater or lower than zero. We mark the vertex whose z* coordinate is zero, as Q_0. Obviously there is only one intersection point between triangle B and H plane, and the point is Q_0, i.e.Q_L=Q_R =Q_0. The projection of Q_L on V surface is Q_{LV}(0,y*$_{Q0}$,0), if Q_{LV} doesn't lie in A_V (this is easy to judge), triangle A separates from triangle B and the test process ends up. Otherwise, we need to judge whether Q_{LW}(x*$_{Q0}$,0,0) lies in A_V, if Q_{LW} doesn't lie in A_W, triangle A separates from triangle B and the test process ends up. Otherwise, we need to have a test, which is a two-dimensional test, whether point Q_L intersect with triangle A_H, if so, triangle A intersects with triangle B at point Q_L, if not, triangle A separates from triangle B. How to judge whether a point intersects with a triangle on a two-dimensional plane are shown in Part III.

4) As is shown in Fig.2 (d), B_V is a line segment or a triangle and has two intersection points with y*-axis. This case will emerge when among the z* coordinates of all triangle B's vertices, one is not equal to zero and the other two are zero. We assume the vertices whose z* coordinates are zero denoted by Q_0 and Q_1. Obviously there are two intersection points between triangle B and H plane and the points are Q_0 and Q_1, i.e.,Q_L=Q_0 and Q_R =Q_1. The projection of Q_L and Q_R on V surface are Q_{LV}(0,y*$_{Q0}$,0) and Q_{RV}(0,y*$_{Q1}$,0). If the line segment L_{BV}(the projection of L_B on V surface) whose endpoints are Q_{LV} and Q_{RV} has no overlap part with A_V, triangle A separates from triangle B and the test process ends up. Otherwise, we need to judge whether L_{BW} has overlap part with A_W, if L_{BW} has no overlap part with A_W, triangle A separates from triangle B and the test process ends up. Otherwise, we need to have a test whether line segment L_B intersect with triangle A_H, if so, triangle A intersects with triangle B at line $L_B \cap A_H$, if not, triangle A separates from triangle B. How to judge whether a line segment intersects with a triangle on a two-dimensional plane is shown in Part III.

5) As is shown in Fig.2(e), B_V has two intersection points with y*-axis. This case will emerge when among the z* coordinates of all triangle *B*'s vertices, one is zero and one is greater than zero and the last one is lower than zero. We assume the vertices whose z* coordinate is zero denoted by Q_0. Obviously there are two intersecting points between triangle *B* and *H* plane, one of the points is Q_0 and the other is Q_R which is the intersecting point between triangle *B's* edge Q_1Q_2 and plane *H*. So we can get Q_L=Q_0 and Q_R =Q_R. The next steps are similar to the steps in 4). How to calculate the intersecting point Q_R are shown in Part III.

6) As is shown in Fig.2(f), B_V has two intersecting points with y*-axis. This case will emerge when among the z* coordinates of all triangle *B*'s vertices, one is greater than zero(lower than zero) and the other two are lower than zero(greater than zero). We assume the vertex whose z* coordinate's sign is different from the other two is Q_0, and the other two are Q_1 and Q_2. Obviously there are two intersecting points between triangle *B* and *H* plane, one of the points is Q_L which is the intersecting point between Q_0Q_1 and plane *H*, the other is Q_R which is the intersecting point between Q_0Q_2 and plane *H*. We can get the points Q_L and Q_R through the algorithm provided in Part III. The next steps are similar to the steps in 4).

III. Algorithm Details

A. Test Whether Two Collinear Line Segments Have Overlap Part

As is shown in Fig.3, there are two collinear line segments L_1 and L_2. First, we get the coordinates of *P,Q,M* and *N*. Second, according to their coordinates, if *Q* is on the left side of *M* or *P* is on the right side of *N*, the lines L_1 and L_2 don't have overlap part, otherwise, they have.

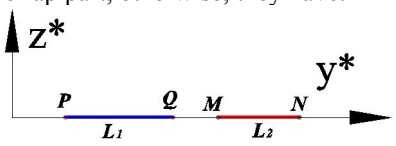

Fig.3 Collinear line segments.

B. Test Whether a Line Segment Intersects with a Triangle on a Two-dimensional Plane

Here is a different area from the original algorithm. The original algorithm uses the algorithm called generalized two-dimentional and three-dimensional clipping proposed by Cyrus-Beck[14] and we use the new algorithm of line clipping for convex polygons put forward by SUN Xie-hua [11] to solve this problem. The algorithm we use is much more efficient than the one used in the original algorithm[11]. First, we calculate the equation of line L_1 as shown in (6). Second, we calculate e_i by putting the coordinates of P_i(i=0,1,2,3;P_3=P_1) into (7). Third, we calculate intersecting points. We use pseudo code form as is shown in Fig.4 to describe the process. Forth, according to the result of the third step, if we get no intersecting point or get only one but it doesn't lie in L_1 or get the intersecting line L_{Q1Q2} but it has no overlap part with L_1, the line L_1 separates from the triangle. Otherwise, they intersect.

$$ax + by + c = 0 \tag{6}$$

$$e_i = ax_i + by_i + c \tag{7}$$

```
int n = 0;//counts of intersection points
Point Q1, Q2, Q;//define two intersection points
float e[4];
if (e[0] == 0){ n++;Q1=P[0];}//P[i]is the vertex of triangle
for (i = 0; i <= 3; i++)
{     if (e[i+1]==0)
      {  n++;
         if (n==1)Q1=P[i+1];
         if (n==2){Q2=P[i+1];break;}}
      if (e[i]*e[i+1]<0)
      {  n++;
         Q=CalInterPoint();//calculate intersect point
         if (n==1)Q1=Q;
         if (n==2){Q2=Q;break;}}}
```

Fig.4 Pseudo code for calculating intersection points.

C. Test Whether Two-Dimensional Triangles Intersect

Here is a correction for the original algorithm. The original algorithm only used the first rule of the following we use to test whether two-dimensional triangles intersect, for the situation as shown in Fig.5(a), it will give the wrong result triangle *A* separate from triangle *B*. So we add the second rule as shown in the following.

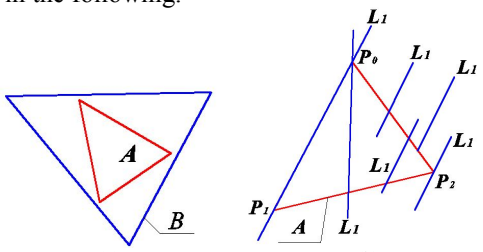

Fig.5 Two-dimensional triangles intersect.

We assume that there are two triangles *A* and B. First, we use the method shown in III-B to judge whether there is an edge of triangle *B* clipped by triangles *A*. All the situations shown in Fig.5(b) the algorithm can deal with. If there is an edge of triangle *B* clipped by triangles *A*, the triangles *A* and *B* intersect[10]. Otherwise, we need to test wheher there is a vertex of triangle *A* lies inside triangle *B*[11]. If not, triangle *A* separate from triangle *B*, otherwise, triangles *A* lies inside triangles *B* as shown in Fig.5(a).

D. Test Whether a Point Intersect with a Triangle on a Two-Dimensional Plane

As is shown in Fig.6 there is a point *P* and a triangle *A*. We translate both *A* and *P* to make *P* lie at the origin. The test becomes that of checking if the origin is contained in the translated triangle. If *P* lies in *A, P* must lie on the same side of all edges of the triangle *A* [12]. That is, either *P* lies to the left of all edges or to the right of all edges. If neither is true, *P* is not contained in the triangle. The implementation is detailed in [12].

E. Calculating the Intersecting Point Between a Line and a Plane

As shown in Fig.1, the edge Q_0Q_1 of triangle *B* intersects with plane *H* at point Q_L,at the same time, line $Q_{0v}Q_{1v}$ (the projection of Q_0Q_1) intersects with y*-axis at point Q_{Lv} [9]. We

can get the parameter equation of points Q_L and Q_{Lv}. Obviously, t is equal to t_v.

$$Q_L = Q_1 + t(Q_0 - Q_1) \tag{8}$$

$$Q_{Lv} = Q_{1v} + t_v(Q_{0v} - Q_{1v}) \tag{9}$$

As shown in Fig.7， we have.

$$t_v = \frac{Q_{Lv} - Q_{1v}}{Q_{0v} - Q_{1v}} = \frac{|Z^*_{Q1}|}{|Z^*_{Q1}| + |Z^*_{Q0}|} \tag{10}$$

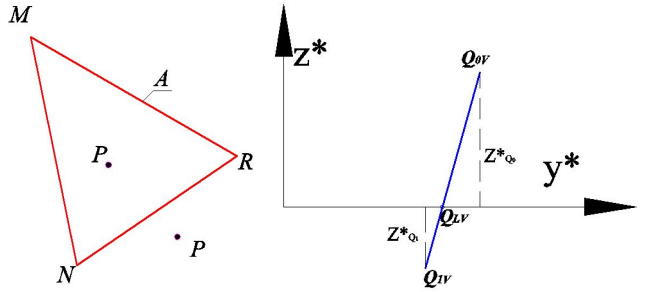

Fig.6 Point and triangle. Fig.7 Projection of edge Q_0 Q_1 on V plane.

IV. Accelerating The Algorithm Using GPU

In order to further enhance the speed of our algorithm, we adopt GPU parallel computing which is not included in the original algorithm.

A. General Instruction

Because triangles aggregates of two objects are independent to each other in collision detection, so we can use the parallel computing ability of GPU to accelerate our algorithm.

B. Detailed Implementation

Here, we will give a brief introduction on how to speed up our algorithm.

1) *Mapping Triangles Vertices Data to Texture Memory*: The texture memory of GPU can store *1*, *2* or *4* components of integer or float vector. We suppose there are two objects marked as O_1, O_2, the number of triangles of O_1 and O_2 are N and M, respectively. We assign two float 4-type 2D texture memories marked *tex1* and *tex2* for O_1 and O_2, the memories size are $N*3$ and $M*3$. At the same time, we assign an unsigned-int-type GPU memory R whose size is $N * M$ and initialize it with 0. Take O_1 as an example, we map the vertices data of the triangle n of O_1 to $tex1(0,n)$, $tex1(1,n)$ and $tex1(2,n)$, so we can get the vertices data of the triangle n of O_1 by the function $tex2D(tex1,j,i)$ ($j \in \{0,1,2\}$ $0 \le i<N$) in kernel.

2) *Mapping Intersection Test to GPU Multi-Thread*: The map between triangle intersection test and GPU multiple threads are shown in Fig.8. We use pseudo code form as shown in Fig.9 to describe this process.

3) *Getting Collision Detection Results*: We copy the array R to host memory through CUDA API, the collision detection result between the triangle numbered m of object O_1 and the triangle numbered n of object O_2 stores in $R[n+m*N_2]$, where N_2 is the number of triangles contained in O_2. If the value of $R[n+m*N_2]$ is *1*, the two triangles intersect; otherwise, the two triangles separate to each other.

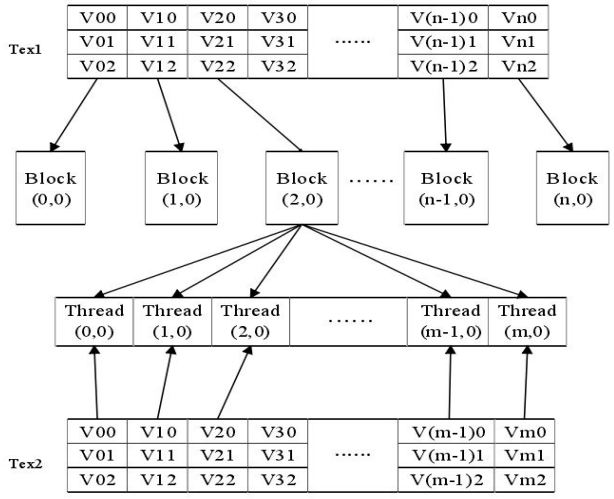

Fig.8 Map between triangle intersection test and GPU multiple threads .

```
//the statement of our algorithm function which can run on GPU
// input：vertices data of the triangles;output：1 or 0
int Tri_Tri_Test3D(float3 P0, float3 P1, float3 P2, flaot3 Q0,flaot3 Q1, float3Q2);
//kernel run on device
// input:a pointer of memory stores the intersection test results
_global_ void kernel(unsigned int* R)
{
int bid = blockIdx.x; // block ID
int tid = threadIdx.x; //Thread ID
int i=threadIdx.x+ blockIdx.x*blockDim.x
float3 P0  = tex2D(tex1, 0, bid); // get vertices data in the texture memory
float3 P1 = tex2D(tex1, 1, bid);
float3 P2 = tex2D(tex1, 2, bid);
float3 Q0 = tex2D(tex1, 0, tid);
float3 Q1 = tex2D(tex1, 1, tid);
float3 Q2 = tex2D(tex1, 2, tid);
  //intersection test results
int R[i] = Tri_Tri_Test3D(float3 P0, float3 P1, float3 P2, flaot3 Q0,flaot3 Q1, float3Q2);
}

//the function who will call the kernel run on host
//N1,N2-- the number of triangles of objects O1andO2
 kernel<<<N1,N2,0>>>(unsigned int* R);
```

Fig.9 Pseudo code for describing the map process .

V. Experimental Results

A. The Experimental Environment

CPU: Inter(R) Core(TM) i3-2100 CPU @ 3.00GHz, Memory:4GB, GPU:NVIDIA GeForce GTS450 with 512MB memory, OS:Microsoft Windows 7 Ultimate, CUDA version:7.0, Development environment: Microsoft Visual Studio2013, Development language:C/C++ and CUDA C.

B. *Detailed Implementation*

1) *To Verify the Performance of the Improved Algorithm*: In order to make our algorithm's every step be executed at least once and test our algorithm's stability, we specially designed 40 triangle pair. We use the original algorithm proposed by Yu Hai-yan [9] and our improved algorithm to test them and the testing process execute 50000 times. That is to say, there are 2 million triangle pairs needed to test. The results are shown in Table I.

Obviously, the improved algorithm improves the efficiency about 41%. The reason lies in the improvement of the original algorithm in this paper: the triangles are classified and handled more reasonable; preclude the situation triangle pair which is separate by use of the projection of triangles in *V* and *W* surface to reduce main calculation; predict the projection of which edge of triangle *B* will intersect with the axis to reduce the unnecessary line intersection tests and avoid solving the singular situation.

TABLE I
PERFORMANCE COMPARISON of THE IMPROVED ALGORITHM and THE ORIGINAL ALGORITHM on CPU

number of triangle pairs(million)	Time consumption (ms)	
	Original algorithm	Our algorithm
2	1256	739

TABLE II
PERFORMANCE COMPARISON of THE IMPROVED ALGORITHM on CPU and on GPU

number of triangle pairs(million)	Time consumption (ms)	
	Our algorithm on CPU	Our algorithm on GPU
2	739	58

TABLE III
PERFORMANCE COMPARISON of THE IMPROVED ALGORITHM on GPU for DIFFERENT NUMBER of TRIANGLE PAIRS

number of triangle pairs(million)	Time consumption (ms)		Time consumption ratio CPU:GPU
	Our algorithm on CPU	Our algorithm on GPU	
0.1	42	26	1.6:1
0.5	231	33	7:1
1	384	41	9.4:1
1.5	557	49	11.4:1
2	739	58	12.7:1

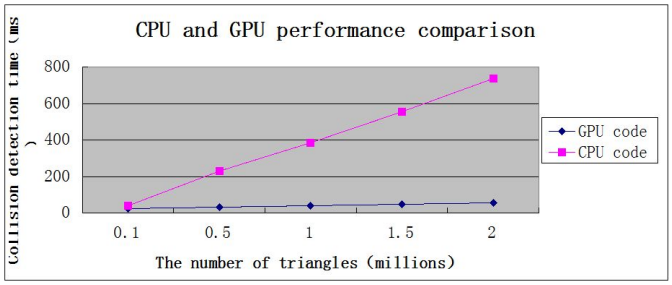

Fig.10 CPU and GPU performance comparison.

2) *To Verify the Performance of the Improved Algorithm on GPU*: We continue to test the above triangle pairs by our GPU accelerated algorithms. The results are shown in Table II. And we test our algorithms on CPU and on GPU for different number of triangle pairs, the results are shown in Table III and Fig.10. From the results we know that the algorithm on GPU is about 13 times faster than the original algorithm on CPU when the triangles' number is 2 million and with the increase of the number of triangles, the CPU simulation showed a linear increase in execution time, and execution time of GPU growth is very slow. This is because the CPU perform all intersecting calculation in order,while the GPU assign the increasing tasks to the multi processor for parallel execute.

So in the test of a large number of triangles,our algorithm has shown a prefect efficiency.

VI. CONCLUSIONS

Our improved method improves the efficiency and correctness of the algorithm proposed by Yu Hai-yan by making a more reasonable and detailed classification on the position relation between the triangle pair and by use of GPU. At the same time, the stability of our improved algorithm is as good as the original. Now it can meet the requirement of practical application.

ACKNOWLEDGMENT

This work is supported by Natural Science Foundation of Heilongjiang Province of China(Grant No.F201416).

REFERENCES

[1] Christer Eticson. Real-Time Collision Detection[M], 500 Sansome Street, Suite 400, San Francisco, CA 94111:Elsevier Inc., 2005. 1-1.
[2] ZOU Yi-sheng, DING Guo-fu, HE Yong, XU Ming-heng. Fast intersection algorithm between spatial triangles[J]. Application Research of Computers,2008,10:2906-2910.
[3] Möller T.A fast triangle-triangle intersection test[J].Journal of Graphics Tools,1997,2(2): 25-30.
[4] HELDM.ERIT: a collection of efficient and reliable intersection tests [J].Journal of Graphics Tools,1997,2(4): 25-44.
[5] TROPP O,TALA, SHIMSHONI I.A fast triangle to triangle intersection test for collision detection[J].Computer Animation and Virtual Worlds,2006,17(5): 527-535.
[6] DEVILLERS O,GUIGUE P. Faster triangle-triangle intersection tests,TR 4488[R]. [S. .l]: INRIA, 2002.
[7] GUIGUE P, DEVILLERS O. Fast and robust triangle-triangle overlap test using orientation predicates[J].Journal of Graphics Tools,2003,8(1): 25-42.
[8] SHEN Hao, HENG P A, TANG Z. A fast triangle-triangle overlap test using signed distances[J].Journal of Graphics Tools,2003,8(1): 3-15.
[9] Yu Haiyan, He Yuanjun. Testing the Intersection Status of Two Triangles[J]. JOURNAL OF GRAPHICS,2013,04:54-62.
[10]HE Da-hua , CHEN Chuan-bo. A Sufficient and Necessary Condition of Two Triangles' Non-Overlapping[J]. MATHEMATICA APPLICATA, 2002,S1:28-30.
[11]SUN Xie-hua. A New Algorithm of Line Clipping for Convex Polygons[J]. Journal of Image and Graphics,2003,12:115-117.
[12] Christer Eticson. Real-Time Collision Detection[M], 500 Sansome Street, Suite 400, San Francisco, CA 94111:Elsevier Inc., 2005. 203-206.
[13] Jason Sanders, Edward Kandrot. CUDA by Example.An Introduction to General-Purpose GPU Programming[M]. Boylston Street, Suite 900 Boston, MA 02116 :Pearson Education, Inc., 2011. 16-153
[14]CYRUS M, BECK J. Generalized two-and three-dimensional clipping [J]. Computers and Graphics, 1978, 3(1): 23-28.