Parallel Collision Detection Algorithm Based on Mixed BVH and OpenMP

Wei Zhao, Rui-pu Tan, and Wen-hui Li

Abstract—**Concerning the requirements of real-time and accurate collision detection in interactive system, we propose a shared memory parallel collision detection algorithm. First we incorporate the merits of both AABB bounding box and bounding spheres to construct a hybrid bounding representation of arbitrary non-convex polyhedra (S-AABB) for attaining speed, and then use OpenMP parallel programming model to traversal the built hybrid bounding volume hierarchy, so further accelerate the collision detection algorithm. At last, experiments results have shown that our algorithm is advantageous over other current typical collision detection algorithms such as I-COLLIDE [1] regarding efficiency and accuracy, so can meet the real-time and accurate requirements in complex interactive virtual environment.**

I. INTRODUCTION

As a classic and critical problem in computer graphics, virtual reality, computer games, animation, computer-aided design, robotics and virtual manufacturing, and other areas [2], collision detection has been more concerned about over the years. At present, more geometric models in virtual reality are composed of thousands of basic geometric elements. With the increase of the geometric complexity of virtual environment, the calculating complexity of collision detection has been greatly increased, and the interaction of complex scenes consumes a large amount of computer resources, so collision detection often becomes a bottleneck in virtual environment.

In recent years, researchers have had extensive and in-depth study on collision detection, and have presented a large number of new efficient collision detection algorithms. These algorithms are mostly divided into two kinds: space divide and BedBox[3]. The aims of the two methods are reducing the numbers of the objects and basic geometrical elements which need to be tested at the most. In particular, the BedBox has been recognized as a relatively good collision detection technology to be widely used. The essential idea of BedBox is using bigger and simple geometrical object (box) to substitute complicated geometrical objects, also gradually approach geometrical characteristic of the object by structuring dendriform hierarchy. When doing the intersect test, we only do farther intersect test on the overlay parts to greatly reduce the number of boxes which participated in the intersect test and enhance the efficiency of collision detection. At present, there are several boxes mostly as follows: AABB (Axis-Aligned Bounding Boxes), Sphere, OBB (Oriented Bounding Box), FDH (Fixed Directions Hulls), and k-dop. Each BedBox has its advantages and disadvantages. In this paper, according the characteristics of BedBox, we present a new parallel collision detection algorithm based on mixed BVH, which incorporates the merits of both AABB bounding box and bounding spheres to construct a hybrid bounding hierarchy (S-AABB).

With the rapid development of parallel computer, parallel technology has brought unprecedented opportunities and challenges for the researchers. Parallel collision detection has been widely applied in computational geometry and robot control and other fields [4]. Reference [4] presented a voxel—based parallel collision detection algorithm, but this algorithm is not fit for the too simple and complex model. Reference [5] gave us a parallel collision detection algorithm based on dividing a convex polyhedron to tetrahedrons, but the test model that is convex has a certain limitations. Reference [6] presented a parallel Octree collision detection based on MPI, although the efficiency of this algorithm has been improved, constructing Octree of the object in this algorithm is considerable time-consuming, and the communication between processes needs to spend a certain amount of time. Reference [7] presented a parallel collision detection algorithm based on pipelining and divide-and-conquer, although the efficiency of this method has been greatly improved, the number of task p has great influence on performance, and it is difficult to find the best value.

Unlike the above methods, we present a parallel collision detection algorithm based on mixed BVH and OpenMP. At first, we build Balance-BoxTree for every object in environment using divide-and-conquer technologies, and then form parallel traversing the task tree. Experimental results have proved that the time efficiency of our collision detection algorithm has been greatly increased. At the same time, our algorithm can run on both single processor computer and multi-processor computer.

II.THEORETICAL FOUNDATION BEHIND OUR ALGORITHM

2.1. The basic collision detection algorithm

The final procedure of collision detection is the accurate

Manuscript received June 20, 2008. This work was supported by the National Natural Science Foundation of China (NO. 60573182，69883004).

Wei Zhao is with the College of Computer Science & Technology, Jilin University, Changchun, Postcode 130012, China (phone:13086846588; e-mail:prince1205@163.com).

Rui-pu Tan is with the School of Computer Science & Engineering, Changchun University of Technology, Postcode 130012, China (e-mail:tanruipu123@163.com).

Wen-hui Li is with the College of Computer Science & Technology, Jilin University, Changchun, Postcode 130012, China (e-mail: wenhuili99@yaho.com).

978-1-4244-1787-2/08/$25.00 ©

intersection detection between the basic geometric elements, and most of the basic geometric elements are triangles. So an efficient triangle and triangle intersection test algorithm is essential to improve the overall efficiency of the collision detection algorithm.

Reference [8] presented two triangle and triangle intersection test algorithms: vector judgement algorithm and scalar judgement algorithm. By comparing the result of simulation, it concludes that vector judgement algorithm based on the improvement and optimization of scalar judgement algorithm is about 7% faster and easier on the same condition. So we use vector judgement algorithm in the narrow phase.

2.2. The hybrid bounding volume hierarchy

In the broad phase, we speed up the algorithm by traversing the hybrid bounding volume hierarchy. Bounding sphere is simple and easy to update after rotating pan, but it also has shortcomings that is the poor close, so need to test more intersection of spheres; In comparison with the bounding sphere, AABB bounding box is closer, and its construction and intersection are simpler. Therefore, we present the hybrid bounding volume hierarchy (S-AABB) which incorporates the merits of both AABB bounding box and bounding spheres, and use it to eliminate the objects and the basic elements which do not intersect apparently in the broad phase.

2.3. The intersection detection between different bounding volumes

Let (x, y, z) be the points in the AABB bounding box, H be the side of the AABB bounding box, (X_B, Y_B, Z_B) be centric point of the AABB bounding box, then the constraint equations of AABB bounding box is:

$$\left\{(x, y, z) \middle| 0 < x, y, z < H \right\} \quad (1).$$

Similarly, let (x, y, z) be the points in the bounding sphere, the radius of the bounding sphere is R_S, the centre locates at the coordinates (X_0, Y_0, Z_0), then the constraint equations of bounding sphere is:

$$\left\{(x,y,z) \middle| (x-X_0)^2 + (y-Y_0)^2 + (z-Z_0)^2 < R_s^2 \right\} \quad (2).$$

We suppose that the AABB bounding box locates at the coordinates origin $(0,0,0)$, set $\varphi = tg^{-1}\left(\frac{Y_0}{X_0}\right)$, where X_0 does not equal to zero, then the distance between the AABB bounding box and bounding sphere is:

$$dist = \sqrt{X_0^2 + Y_0^2 + Z_0^2} - (R + H) \bullet \varphi \quad (3).$$

Let the angle between the line which link the center point of the AABB bounding box to the center point of the bounding sphere and the Z coordinate axis be θ, then

When Y_0>0 and X_0>0 $\theta = \varphi$;

When Y_0>0 and X_0<0 $\theta = \pi - \varphi$;

When Y_0<0 and X_0>0 $\theta = -\varphi$;

When Y_0<0 and X_0<0 $\theta = \pi + \varphi$;

Therefore, according the comparison between $\theta \bullet dist$ and $|Z_0|$, we can obtain the results of the intersection; if $\theta \bullet dist < |Z_0|$, then the intersection between the AABB bounding box and the bounding sphere happens. Otherwise, it does not happen.

2.4. BalanceTree and divide and conquer [7], [9]

2.4.1. BalanceTree. We define Bintree to be full just because BoxTree's leafnode is a single element subclass of S. Based on that expression there are $2^n - 1$ nodes at most in the BoxTree and n of those are leafnode, also each two childnode has the same number of boxes. The height of a full Bintree is $\log 5n$ at least, then it is called BalanceTree.

2.4.2. Divide and conquer. Divide and conquer is a basic technology in parallel programming. Its characteristic is dividing the problem into several same and small problems until the problem cannot be divided. As the dividing going, the recursive model will form a Bintree. It is also applied to divide one task into M ($M > 2$) parts and called M-divide and conquer. There are Quartotree ($M = 4$) and Octaltree ($M = 8$).

III. ALGORITHM DESCRIPTION AND IMPLEMENTATION

3.1. Constructing the hybrid bounding volume hierarchy

Given a set of polygons of an object, the construction of the hybrid bounding volume hierarchy is a four-pass process as follows:

(1) we construct a small bounding sphere for each polygon of the object with a high efficiency bounding sphere generation algorithm which is presented in [10].This sphere is named "underlying sphere" and serves at least two purposes: 1. to help fast subdivision of the bounding volumes. 2. to be used as a pre-check aid in the fundamental algorithm.

(2) The root node of the hierarchy is built. The root node mainly consists of the all underlying spheres, an axis aligned bounding box of all of the polygons, and a bounding sphere of all of the polygons.

(3) The root node is subdivided into two sub-nodes which have the same attributes as those of the root node. This is accomplished by: 1. The bounding box of the root node is divided into two sub-boxed by an axis aligned dividing plane; 2. Those underlying spheres fully covered by the left or right sub-box are considered to be contained in the corresponding left or right sub-node. Those underlying spheres which cross both sub-boxed are included in both left and right sub-node. 3. The

construction of bounding spheres of the sub-nodes takes almost the same steps as that of the root node, but only those polygons contained in the corresponding sub-node are considered.

To make an approximately balanced binary tree which is characterized by the number of underlying spheres contained in each sub-node, and to minimize the number of those underling spheres which cross both sub-boxes, we adopt a penalty function $f(p)$ to determine the dividing plane:

$$f(p) = |n_l - n_r| + \lambda n_c \quad (4)$$

Where p is the axis aligned dividing plane, n_l and n_r are the number of underlying spheres fully covered by the left and right sub-boxes respectively, n_c is the number of those underlying sphere crossing both left and right sub-boxes and λ is a constant. Ideal dividing plane is the plane whose value of the penalty function in all axes is minimum:

$$\min\left\{\begin{array}{l}\min\{f(p)\,|\,p_x \perp x\_axis, p_x \in [x_{\min}, x_{\max}]\} \\ \min\{f(p)\,|\,p_y \perp y\_axis, p_y \in [y_{\min}, y_{\max}]\} \\ \min\{f(p)\,|\,p_z \perp z\_axis, p_z \in [z_{\min}, z_{\max}]\}\end{array}\right\} \quad (5)$$

(4) The same subdivision procedures described in the above step are recursively applied to both newly generated sub-nodes to acquire the final hierarchical hybrid bounding volume representation of the object.

Appropriate exits are crucial for a recursive procedure, so we use linear function to make proper recursive terms:

$$p(x) = A^T * B \quad (6)$$

notation: $A = (a1,a2,a3,...an)$ $B=(b1,b2,b3,...bn)$. *b1,b2,b3,...*bn are the elements of recursive finish, they are always as follows: the height of BoxTree, the number of elements in leafnode. *a1,a2,a3,...an* are the relevant maces.

Fig.1 is the resulting approximately balanced binary tree, where the two numbers in the first row of each node represent respectively the number of underlying spheres (hence polygons) contained in the node, and the three numbers in the second row of each node represent the width, height and depth of the bounding box of the node.

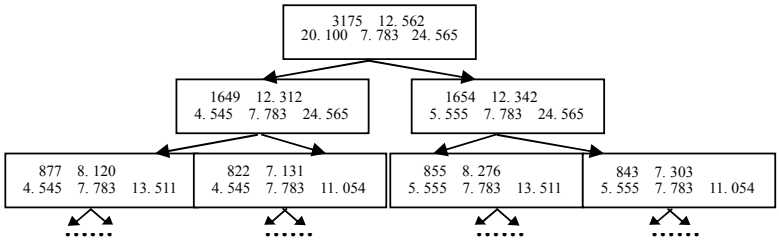
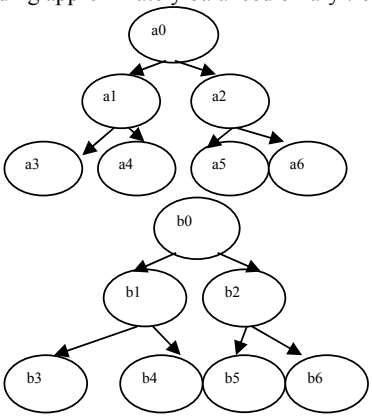


Fig 1. Representation of the hybrid bounding volume hierarchy

3.2. Traversal of the hybrid bounding volume hierarchy

Once the hybrid bounding volume hierarchies of two objects A and B are constructed, collision detection between the two objects can be performed by first traversing the corresponding approximately balanced binary trees of A and B. So we introduce the concept of task tree [7]. Then the procedure of traversing the corresponding approximately balanced binary trees of A and B simultaneously can be regarded as the traversal of a task tree. Fig.2 illustrates a task tree.



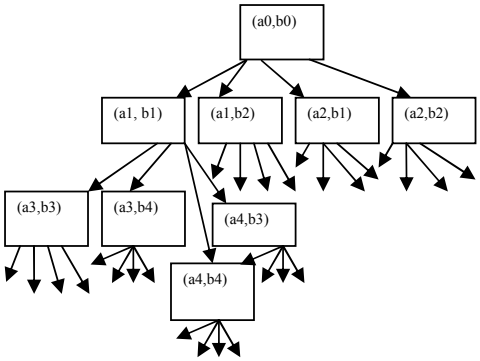


Fig.2 Simple Work-Tree

The collision detection process is in nature an or-tree search process of the task tree. On the one hand, if collision is found at some subtask, then collision occurs between the two objects A and B. On the other hand, if no collision can be found after all the subtasks are processed, then no collision occurs between objects A and B. So the collision detection result of each task can be achieved by the logical OR of the collision detection result of its all subtasks. Taking advantage of task tree, the Width-First-Search-based traversal of the hybrid bounding volume hierarchy can be given as follows

```
Input: root node of the task tree (RA,RB)
Output: whether A and B collide
        //TRUE: collision, FALSE: no collision
Bool WFS-Traversal (RA,RB)
{
   LIVESET<-root task node (RA,RB)
   //push the root node in the queue
   for i =1 to LIVESET.size
    {
     while (LIVESET!=NULL)
       (a1,b1)<- LIVESET;
       //pop a task node up from queue
       SplitNode=true;
       if (not BoundingSphereOnerlap(a1,b1)){
          SplitNide=FALSE;
          //no collision, hence no subtasks
       else //bounding spheres do intersect
          Flag1=(Sphere2BoxVolumeRatio(a1)>=
                 RatioThreshold(a1) );
          Flag2=(Sphere2BoxVolumeRatio(b1)>=
                 RatioThreshold(b1) );
        if (Flag1)
          //bounding box of a1 should be respected
          if (Flag2)
          //bounding box of b1 should be respected
          if (not BoundingBoxsOnerlap(a1,b1))
            {
             SplitNode=FALSE;
          endif // bounding box not overlap
       else// Flag2=FALSE, bounding box of b1 ignored
       if(notBoundingBoxsBoundingSphereOnerlap(a1,b1))
          SplitNode=FALSE;
       endif//bounding box and bounding sphere not
            overlap
      endif // flag2
      else // Flag1=FALSE, bounding box of a1 ignored
       if(Flag2){
        //bounding box of b1 should be respected
       if(notBoundingBoxsBoundingSphereOnerlap(b1,
       a1))
          SplitNode=FALSE;
       endif//bounding box and bounding sphere not
          overlap
        endif // flag2
       endif // flag1
      endif //not BoundingSphereOverlap(a1,b1)
      if(SplitNode=FALSE)//no collision, hence no subtasks
        free the memory occupied by this code;
      else //create subtasks or do collision detection by
             adopting fundamental algorithm
        if (IsLeafNode(a1)) //a1 is a leaf
         if (IsLeafNode(b1)) //b1 is a leaf

        CollisionDetectionFundamentalAlgorithm(a1,b1);
         if (a1 and b1 collide)
            return TRUE;
          endif
         else
            LIVESET<- (a1,b1-> left);
            LIVESET<- (a1,b1-> right);
         endif // IsLeafNode(b1)
        else // a1 is not a leaf
         if (IsLeafNode(b1))
          {
           //b1 is a leaf, two subtasks need to be created
            LIVESET<- (a1-> left,b1);
            LIVESET<- (a1-> right,b1);
        else
            LIVESET<- (a1-> left,b1-> left);
            LIVESET<- (a1-> left,b1-> right);
            LIVESET<- (a1-> right,b1-> left);
            LIVESET<- (a1-> right,b1-> right);
          endif // IsLeafNode(b1)
         endif // IsLeafNode(a1)
        endif // SplitNode
       endwhile //LIVESET
      endfor
     return FALSE; // A and B not collide
   }
} //WFA-Traversal
```

Where, the queue, LIVESET, is used to store the task nodes which need to be processed. The threshold what the function RatioThreshold() returns for the volume ratio is associated with the level of current node in its bounding volume hierarchy, and is determined in a statistical sense after all the volume ratios of its sibling nodes are taken into account.

IV. PARALLEL ALGORITHM BASED ON SHARED-MEMORY OPENMP

4.1. Knowledge related OpenMP [11], [12]

OpenMP is a parallel programming model based on the threads, is shared programming model based on the existed threads, is an external programming model, and it enable programmers to fully control the parallelism.

OpenMP uses the parallel implementation model of Fork-Join. The OpenMP program begins in a single main thread (master thread). The main thread will be carried out by serial form until it encounters the first parallel domain, then is carried our by parallel form. The implementation process as follows: 1. Fork: the main thread creates a team of parallel threads, and then, implements parallel domain codes in different threads; 2. Join: When the implementation

of the main thread in the parallel jurisdictions finishes, else threads will be suspended simultaneously, at last, only the main thread in implementation.

4.2. Parallel collision detection algorithm

Parallel regions are more tied to loop constructs in the proposed algorithm, these loops can efficiently be executed in parallel, to reduce the computation time. So the notes in the queue can be implemented by parallel form, which can improve efficiency of the algorithm. The parallel collision detection algorithm based on OpenMP is as follows:

```
Input: root node of the task tree (RA,RB)
Output: whether A and B collide
        //TRUE: collision, FALSE: no collision
Bool Parallel-Traversal (RA,RB)
{
  LIVESET<-root task node (RA,RB)
  //push the root node in the queue LIVESET
   while (LIVESET!=NULL)
    // the queue LIVESET is not null
      (a1,b1)<- LIVESET;
      //pop a task node up from queue
     #pragma omp parallel for

     if (the current task finds there is no intersection)
        cancel current task ;
     else
     if (there is a leaf node in the current task)
      {
        CollisionDetectionFundamentalAlgorithm();
        if (the result is TRUE)
           cancel all the other tasks being parallel
           processed
           return TRUE;    // collision happen
        endif
        else
           push subtasks into the queue LIVESET in a
           similar manner;
      endif
     endif
     the main thread waits for the completion of all other
     tasks parallelly processed.

     # pragma omp critical
        StoreIntersectingTaskNodes();
   endwhile
  return FALSE;
}
```

V. THE EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

This algorithm is realized at four R10000 processor computer which can execute 64 threads and we designate the number as 32. We use OpenMP parallel programming model to achieve our parallel collision detection, in which partial parallel realization used multi-thread to share the same address space.

Experimental environment is a simple dynamic scenes, there are 31 random moving objects in the environment, the basis geometrical elements are over 80 000. To illustrate the superiority of our algorithm, we do the following three experiments.

Experiment 1: the analysis of time complexity

In the experiment, we compared our algorithm with other classical algorithms from three aspects of the time complexity, frame rate and the total time run 1 000 steps. The experimental results are shown in Table 1 and Fig.3 (a).

Table 1. The analysis of time complexity

Type of Collision	Time Complexity	Frame Frequency (f/s)	Total Time Run 1000 Steps (sec.)
BCDA	$O(n^2)$	3.19	5109
I-COLLIDE	$O(n \log n)$	4.05	4231
SPHERE	$O(n \log n)$	4.05	3265
SCDA	$O(n \log n)$	7.05	185
PCDA	$O(n \log n)$	17.70	74

Where BCDA (Base Collision Detection Algorithm) stands for triangle and triangle intersection test algorithm; I-COLLIDE stands for the collision detection algorithm based on AABB bounding box; SPHERE stands for the collision detection algorithm based on bounding sphere; SCDA (Serial Collision Detection Algorithm) stands for the collision detection algorithm based on mixed BVH; PCDA (Parallel Collision Detection Algorithm) stands for the parallel collision detection algorithm based on mixed BVH and OpenMP.

Can be seen from the results, comparing to those algorithms which use single bounding box, the time efficiency of our algorithm has been greatly improved. The frame rate of PCDA algorithm up to 17.70 / s, and is 3 to 6 times faster than other algorithms, but the total time run 1 000 steps is 1 / 100 to 1 / 3 of the total time of other classical algorithms.

Experiment 2: the analysis of parallelism

To illustrate the advantage of parallel computing, in the given experimental environment, we applied OpenMP to parallel BCDA、I-COLLIDE、SPHERE、SCDA. The experimental results are shown in Table 2 and Fig.3 (b).

Table 2. The analysis of parallelism

Type of Collision	Serial and parallel algorithms	Total Time Run of 1000 Steps (sec.)
BCDA	NP(Non-parallel)	5109
	P(parallel)	2335
I-COLLIDE	NP(Non-parallel)	4231
	P(parallel)	1666
SPHERE	NP(Non-parallel)	3265
	P(parallel)	1336
SCDA	NP(Non-parallel)	185
	P(parallel)	74

Can be seen from the results, in the given experimental environment, using OpenMP parallel programming model, the running time of each algorithm is 2 to 3 times less than their serial algorithms, and our parallel algorithm PCDA is 2.5 times faster than our serial algorithm. Parallel technology further accelerates the process of collision detection.

Experiment 3: the test of performance and efficiency

The test of the performance of our parallel algorithm is mainly to test the parallel efficiency and speedup, we can adopt (7) and (8) to complete.

$$S_{(p)} = \frac{t_s}{t_p} \tag{7}$$

Where $S_{(p)}$ stands for the speedup, t_s stands for the total time of the algorithm runs on single processor, and t_p stands for the total time of the algorithm runs on multi-processor which has p processors.

$$E = \frac{t_s}{t_p \times p} = S_{(p)} \times \frac{1}{p} \tag{8}$$

Where E is the parallel efficiency, p is the number of processors. In other words, efficiency can be defined as the ratio between speedup and the number of CPU.

In the given experimental conditions and circumstances, we test speedup and parallel efficiency when the number of the processors is not same. In the experiment, the speedup of our parallel algorithms is $S_{(p)}$ =2.697, the parallel efficiency is E =0.674. The results are shown in Fig. 4 (c), Fig. 4 (d).

Can be seen from the experimental results, with the increase in the number of processors, the speedup shows a downward trend after reaching the peak, and the parallel efficiency has been declining. Under normal circumstances, 2 to 4 processors will be sufficient in the complex virtual environment.

Parallel process significantly improved the speed of the collision detection algorithm, so can meet the real-time and accuracy request of the interactive virtual environment.

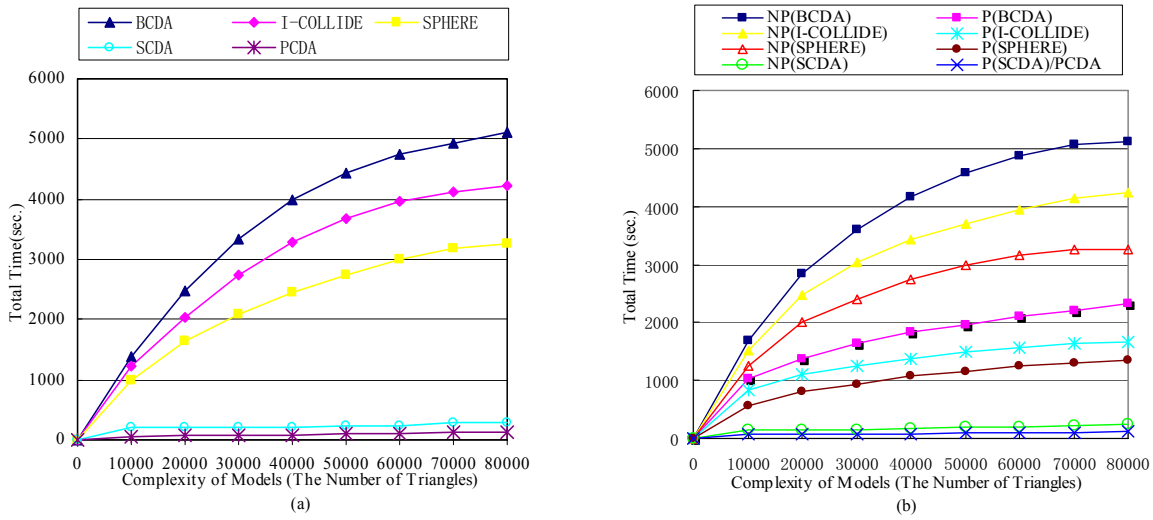
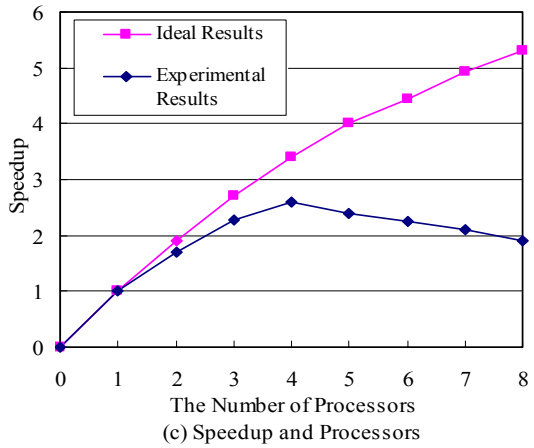
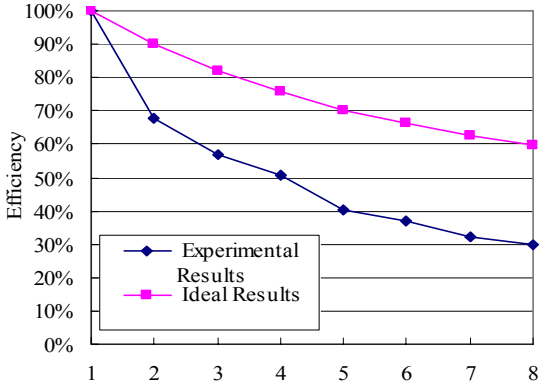


Fig. 3 (a) Compare about experiment data of the five algorithms such as BCDA I-COLLIDE SPHERE SCDA and PCDA run 1000 steps; (b) Compare about experiment data of the four algorithms such as BCDA I-COLLIDE SPHERE SCDA and their parallel algorithms run 1000 steps.



(c) Speedup and Processors



(d) Efficiency and Processors

Fig. 4 (c) The relationship between speedup and the number of processors. (d) The relationship between parallel efficiency and the number of processors.

VI. CONCLUSIONS

We propose a quick parallel collision detection algorithm based on mixed BVH and OpenMP, its characteristics are as follows:

(1) Taking advantage of the merits of both AABB bounding box and bounding spheres to construct a hybrid bounding hierarchy. In the broad phase, through the intersection detection between bounding sphere and bounding sphere, bounding sphere and bounding box, bounding box and bounding box, we rule out a large number of polygons which will be not intersect apparently.

(2) Using divide and conquer in setting up Balance-BoxTree to make sure each subtask has the same size, we can make full use of the multi-processor performance, and also proposed linear function for dividing box.

(3) Applying multi-threading in traversing.

(4) Our algorithm can realize the real-time collision detection in a dynamic and complex scenes as well as interactive virtual environment.

Nevertheless, there is still some room for the improvement. Future work will be focused on:

(1) The algorithm will be better parallelized by the parallelization of intersection tests between bounding boxes and so on.

(2) Using M-divide and conquer in setting up Balance-BoxTree;

(3) Choosing proper maces of a1,a2,a3,…an;

(4) Applying PVM (parallel virtual machine) in order to execute this algorithm in Internet;

(5) We can also use new parallel technology to expand our algorithm, such as MPI, symmetry breaking technology.

ACKNOWLEDGMENT

The authors would like to thank the National Natural Science Foundation of China (NO. 60573182, 69883004) for the financial supports.

REFERENCES

[1] Cohen J, Lin M, Manocha D, Ponamgi M. I-COLLIDE: An interactive and exact collision detection system for large-scale environments, *In: Proc. of the ACM Interactive 3D Graphics Conf*, pp.189~196, 1995.

[2] Fan ZhaoWei, Wan Huagen, Gao Shuming. Streaming real time collision detection using programmable graphics hardware, *Journal of Software*, vol.15, no.10, pp.1505-1514, 2004.

[3] Stephen J Adelson, Larry F Hodges. Generating exact ray-traced animation frames by reprojection, *IEEE Compute Graphics and Applications*, vol.15, no.3, pp.43-52, 1995.

[4] Lawbr Orion Sky, Kal'e Laxmikant V. A voxel—based paralel collision detection algorithm[C], *Proceedings of the 2002 International Conference on Supercomputing,* New York, pp.285-293, 2002.

[5] Xue Guangtao, Li Chao, You Jinyuan. Algorithm of Parallel Collision Detection Based on Dividing a Convex Polyhedron to Tetrahedrons, *Journal of Shanghai Jiaotong University*, vol.38, no.8, pp.1385-1388, 2004.

[6] Liu Xiaoping, Cao Li. Parallel Octree Collision Detection Based on MPI, *Journal of Computer-Aided Design & Computer Graphics*, vol.19, no2, pp.184-187+192, 2007.

[7] Zhao Wei, He Yanshang. Rapid algorithm for parallel collision detection, *Journal of Jilin University (Engineering and Technology Edition)*, vol.38, no.1, pp.152-157, 2008.

[8] Xu Qiang, Lu Xiaofeng, Ma Dengwu. A Survey of Triangle and Triangle Intersection Test, *Computer Simulation*, vol.08, no.23, pp.76-78+145, 2006.

[9] Fan Zhaowei, Wan Huagen, Gao Shuming. A Parallel Algorithm for Rapid Collision Detection, *Journal of Systerm Simulation*, vol.9, no.12, pp.548-552, 2000.

[10] Ritter J, An efficient bounding sphere, *in Graphics Gems (edited by Andrew Glassner),Acaemic Press,Inc,*1990, pp.301-303.

[11] Mauro Figueiredo, Terrence Fernando. An efficient parallel collision detection algorithm for virtual prototype environments. *Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on Volume*, Issue, 7-9 July 2004, pp.249 – 256.

[12] Chen Guoliang. *Design and Analyse on Parallel Arithmetic(revised edition)*. Beijing: Higher Education Press, 2000.