Appear in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2006

A Time-Budgeted Collision Detection Method

Yu-Te Lin
Computer Science Department
Stanford University
yutelin@stanford.edu

Tsai-Yen Li
Computer Science Department
National Chengchi University
li@nccu.edu.tw

***Abstract* - Collision detection is a critical module in many applications such as computer graphics, robot motion planning, physical simulation, CAD/CAM, and molecular modeling. Many efficient algorithms have been proposed to solve the collision-detection problem, and most of them use some sort of hierarchical bounding volume to speed up the time-consuming process. In this research, we focus on a special situation for applications such as interactive virtual environment, where accuracy may not be crucial but the available time for performing the check is limited. We will describe our findings on the inherencies of bounding volume traverse trees (BVTT). Based on this observation, we will propose a time-budgeted collision detection method that can traverse the critical regions of BVTT earlier if the time budget is limited. Preliminary experimental results are also reported, and a new strategy is suggested based on the concept of confidence value.**

Index Terms – Collision Detection. Bounding Volume Hierarchy. Time-Budgeted Algorithms.

I. INTRODUCTION

Collision detection is widely used in computer graphics, robot motion planning, physical simulation, visualization, manufacturing, and molecular modeling. In addition, the collision-detection routine is usually the most time-consuming module in these applications. An intuitive solution to the basic collision-detection problem is by decomposing the objects into geometric primitives and testing all primitive pairs between two objects. However, in most situations, only a few pairs of primitives are really close to each other and many others are far apart. Therefore, many collision detection algorithms tend to cover the primitives with bounding volumes of various sizes organized in a hierarchy and check overlaps between bounding volumes. By doing so, most checks on primitive pairs can be eliminated, and consequently the collision detection process can be significantly sped up. Most efficient collision detection algorithms take this approach but just the types of bounding volumes are different.

There are always tradeoffs between space complexity and time complexity. It is not easy to find a general solution that works in all situations. Therefore, in this work, we focus on a special problem where the time budget available for a collision check is limited. For example, in the application of virtual environment, the accuracy of the check result can be sacrificed if necessary. For such a problem, if we spend all available time on detecting but the procedure could not be completed in time, we generally get nothing from this operation. The time-budgeted collision detection method proposed in this paper uses some simple data structures to record the critical process of previous detections in order to provide information for the applications to predict the probable answer. This method is most suitable for real-time virtual environment systems which may not require a complete collision result but a prompt one. Most importantly, they tolerate some minor errors. The proposed method makes use of some idea of [1] which assumes that the features between two objects change gradually in a continuous path. In [2] the authors report that spatial coherence can be observed in the bounding volume traverse tree (BVTT) such that the list of checked nodes could be re-used in consecutive tests in a continuous path. We have done some more experiments to further analyze the features of a single BVTT and the changes between two consecutive BVTT's. Based on the findings we introduce a best-first principle for deciding how to branch in a BVTT and the concept of confidence value for predicting the answer directly if the time limit is reached. We have implemented our ideas and modified the PQP collision detection package [3], a sibling of the prevalent RAPID [4], to perform experiments.

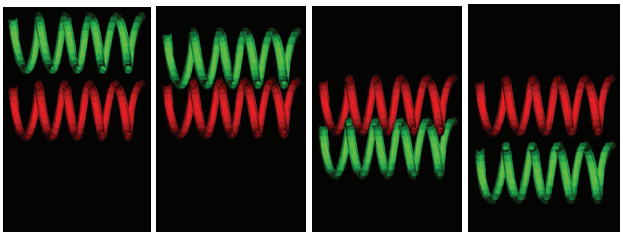
The rest of this paper is organized as six sections. Section II describes the related work in collision detection. Section III presents the discovery of bounding volume coherence and BVTT's shapes. Section IV explains the best-first principle for branching and predicting check result. Section V shows the details of implementation and experimental results. Section VI discusses possible future work and summarizes this research.

II. RELATED WORK

Collision detection is the most critical problem in many applications, and attracted the attentions of many researchers. However, although many prominent algorithms have been proposed; there is no general good solution for all collision detection problems. When object motions are considered, the algorithms could be categorized into *static* methods and *continuous* methods. The static methods check the overlap of objects in one single configuration, while continuous methods check a list of continuous configurations along a path. Below, we will briefly review the most popular methods that have been proposed in the literature.

A. Bounding Volume Hierarchies

Most efficient approaches build a bounding volume hierarchy for each object in pre-computation. Every node in a hierarchy usually represents a simple bounding volume enclosing the underlying geometry. For example, in the oriented bounding boxes (OBB) approach [3][4], a node with a center,



(a) initial (b) first contact (c) last contact (d) goal

Fig. 1 Experimental scene with two 1800-polygon springs acting as the robot and the obstacle, respectively

a rotation, and three dimensions is used to describe a bounding volume. Every node covers the geometry enclosed by all the nodes in its sub-trees. Therefore, for OBB, the root node is the biggest box containing the whole object. By comparing the BV's of two objects and recursively traversing their BVH when they overlap, we can quickly discard safe portions and focus on the overlapped regions. So far, many kinds of BVs have been proposed, such as sphere trees [5], axis-aligned bounding box (AABB) [6], oriented bounding box (OBB), discretely-oriented polytopes (k-DOPs) [7], and swept sphere volumes (SSVs) [3]. From [8] the trade-off between the tightness of models and the complexity of comparing operations has been discussed. The conclusion is that there is no single BV that works the best for all circumstances. For collision checks in a continuous path, the BHV methods usually segment a path with a resolution value ε and check a sequence of discrete configurations instead.

B. Feature tracking, Swept volume and adaptive bisection check

1) Feature tracking methods, such as Lin-Canny method [1] and V-Clip [9], make use of the fact that the closest features change locally in a continuous path. Whenever a closest feature is changed, they can find a new one in linear time. This method requires that the models be built with convex components. However, decomposing a non-convex model into convex parts is a difficult problem which limits the practicability of this approach.

2) Swept volume intersection [10][11] methods are mostly used in continuous checks. They compute the swept volume of the motion and test the volume for intersection. Although exact and complete, this method may lead to a more conservative result since the swept volume represents the geometry for the whole time span. However, although the time dimension helps to improve exactness, the computation complexity also increases, and almost no pre-computation can be adopted to speed up dynamic volume calculation.

3) Adaptive dynamic bisection check [12][13] method contain two main ideas. First, it relates configuration changes to path lengths in workspace. This operation is to get the upper bound lengths of a configuration transition. Second, it uses distance computation rather than real collision detection. Actually, it will get two lower bound distances form the old and the new configurations to the same obstacle. Then, it uses a simple inequality of triangle to test the overlap. If an overlap is found, it will recursively bisect the path and check each segment. This method is especially useful for robot arm planning.

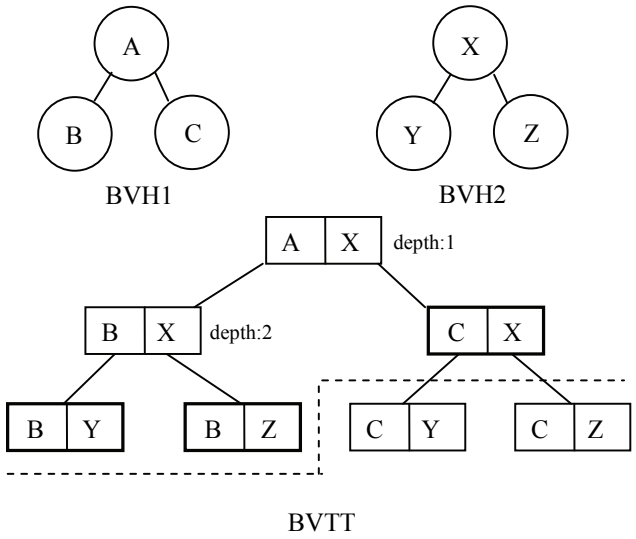


Fig. 2 A sample BVTT formed by BVH1 and BVH2. BVTT's root is a combination of the roots from each BVH. The profile cut (depicted by dotted line) represents the list of lowest nodes that are traversed. The average depth of BVTT in this example is (1+2+2+3+3)/5 = 2.2

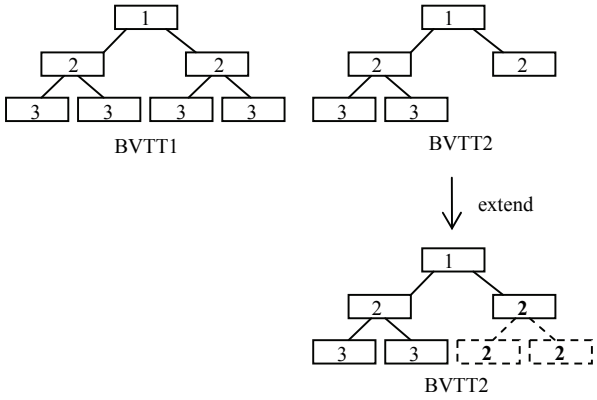


Fig. 3 Computing BVTT differences. BVTT2 (3, 3, 2) is extended to BVTT2' (3, 3, 2, 2) for comparison. BVTT1's leave depths are (3, 3, 3, 3), and therefore the difference between BVTT1 and BVTT2 is (0, 0, 1, 1).

III. Features of Bounding Volume Traverse Trees

In order to observe the characteristics of bounding volume checks in continuous collision detection with BVH, we set up a simple experimental environment containing two models of 1800-polygon spring for the robot and the obstacle, respectively, as shown in Fig. 1. The obstacle is placed between the initial and goal. We monitor the robot moving from a collision free configuration segment through a collided segment, and then to another collision free segment again.

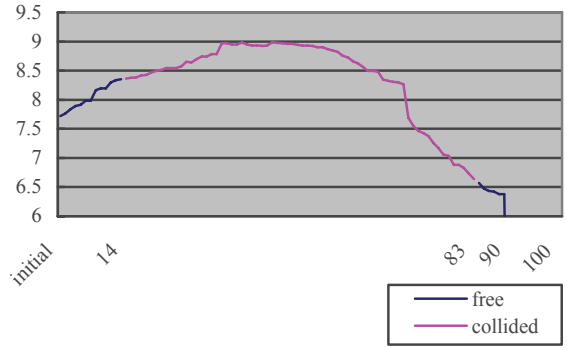


Fig. 4 Average depth diagram of 100 detections. Initial to 13 and 84 to 100 are the collision-free periods. 14 to 83 is the in-collision period.

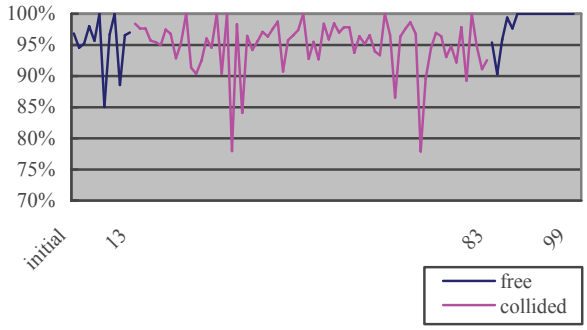


Fig. 5 Zero-difference percentages for 99 samples

We have done some experiments to monitor the traversal behavior of BVTT in performing collision checks. We define some frequently used terms as follows. First, a node of BVTT represents a combination of two nodes from each BVH; and the depth of a node means its distance from the root, as shown in Fig. 2. Initially, the depth of the root node is set to one, and its directly connected children have the depth of two. The average depth for a collision check is the average of all checked nodes' depths in a run. In the example of Fig. 2, the dashed line separating the checked and unchecked nodes is called *profile cut* and the nodes right above the cut is called *cut list*. Second, the depth difference between two nodes is the difference of their depth value as depicted in Fig. 3. In order to compare two cut lists, the higher level nodes will descend to the same level as the lower one. In the example of Fig. 3, the depth difference is (0, 0, 1, 1).

A. *Inherency between consecutive BVTTs*

In the monitoring experiment, the path was divided into 100 segments. By testing these 100 continuous configurations, we can compare the average depths of the collision-free ones to the in-collision ones. In this experiment, the deepest node has the depth of 27, but the average depth ranges from 6.6 to 8.9 for the collision-free cases and 6.5 to 8.3 for the in-collision cases. Actually, we also had observed that the average could be as low as one in some extreme cases. Therefore, no apparent relations have been observed by comparing the number of average only.

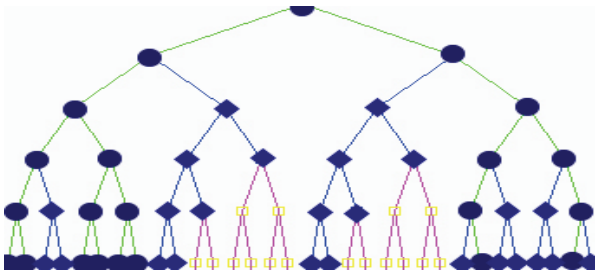


Fig. 6. Part of BVTT. Round nodes are in-collision, diamond nodes are checked free, and the rest are free without further checks.

Nevertheless, if the entire process is taken into account, some clues on the relations can be observed. Fig. 4 depicts the depth averages for all configurations in the path. The collision segment starts from the 14th to the 83rd configuration and the collision-free segments start from the initial to the 13th and from the 84th to the 100th configurations. Therefore, a trend of increasing and then decreasing average depths could be observed when the robot is getting closer to and then getting farther from the obstacle. In addition, the changes are continuous.

We also compare the percentages of zero differences for every two successive BVTT's. In Fig. 5, we depict the values of zero-difference percentage for 99 pairs along the path. We observed that the values are all very close to 100%, which show that significant temporal cohesion exists on the cut patterns of BVTT's for two consecutive configurations. Therefore, from the data of Fig. 4 and Fig. 5, we can summarize that two successive collision checks with an appropriate resolution have similar behavior in traversing their BVTTs.

B. *Shapes of BVTT Cut List*

Although it is somewhat brute-force to analyze the BVTT's of all the configurations in a path, it is useful to monitor the patterns of how the BVTT's are traversed when the configurations are collision-free or in-collision. In this research we have designed a tool to visualize the profile cut in all BVTT's. In general, the BVTT's are very large and require several thousand square pixels' areas for display on screen. Therefore, in Fig. 6, we only show a small part of a typical BVTT taken from our experiment. With this tool, we have found that if a robot collides with an obstacle, the profile cut of its BVTT will contain several wave peaks, and the nodes that are in-collision usually lie in the wave trough. Moreover, the most frequently checked region is usually near the trough. According to this observation, we have made the following rational assumption: if two objects collide, their BVTT will have one or more trough region(s) in-collision.

IV. TIME-BUDGET COLLISION DETECTION

Algorithm BASIC_BV_COLLISION(Node *N1*,Node *N2*)

1. **if N1** and **N2** are leaves
   return CHECK_PRIMATIVE_COLLISION(*N1*, *N2*)
2. **if** *N1* is larger than *N2*
   2.1 **for each** child *N1[i]* of *N1*
      2.1.1 **if** CHCEK_BV_COLLISION(*N1[i]*, *N2*) is *collision*
      2.1.2 **return** *collision*
3. **else** // *N1* is smaller than *N2*
   3.1 **for each** child *N2[i]* of *N2*
      3.1.1 **if** CHECK_BV_COLLISION(*N1*, *N2[i]*) is *collision*
      3.1.2 return *collision*
4. **return** *no collision*

Fig. 7. Original BV recursive check algorithm.

Algorithm PRIORITY_BV_COLLISION(Node *N1*,Node *N2*)

1. **if** *N1* and *N2* are leaves
   return CHECK_PRIMATIVE_COLLISION(*N1*, *N2*)
2. **if** *N1* is larger than *N2*
   2.1 **sort** *N1*'s child nodes to *N1[i]* with priority
   2.2 **for each** *N1[i]*
      2.2.2 **if** CHCEK_BV_COLLISION(*N1[i]*, *N2*) is *collision*
      2.2.3 **return** *collision*
3. **else** // *N1* is smaller than *N2*
   3.1 **sort** *N2*'s child nodes to *N2[i]* with priority
   3.2 **for each** *N2[i]*
      3.2.1 **if** CHECK_BV_COLLISION(*N1*, *N2[i]*) is *collision*
      3.2.2 **return** *collision*
4. **return** *no collision*

Fig. 8. BV recursive check algorithm with priority in branching.

In the process of detecting collisions with BVH's, whenever the bounding volumes of two objects collide, one volume is usually chosen to break into two smaller bounding volumes. The bounding volume check routine will be recursively called on these two smaller volumes and the volume of the other object. Since this recursive algorithm usually traverses the BVTT in a depth-first fashion, choosing which one to branch may have a great influence on the overall efficiency in finding the collided node. For example, in PQP, they make use of the heuristic that bigger volumes will result in higher possibilities in collision. Nevertheless, as shown in Fig. 7, although most BV algorithms will choose the object with a larger volume to break, they do not impose priority on the order of its children. If we have more information about each node, we can have a better way to prioritize the child nodes for a better checking sequence and find the collided geometry with less time. We proposed to modify the algorithm with this prioritized branching as shown in Fig. 8. Moreover, if no enough time is allowed for traversing the whole BVHs, no certain answer about the collision status may result. In this case, the time-budgeted collision detection method will play an important role in providing probable answer to the result.

Algorithm COMPUTE_CONFIDENCE

1. **if** number of checked BV < max number of allowable BV
   1.1 **return** *100*
2. **else**
   2.1 Get the percentage (*r*) of checked nodes in the dangerous regions
   2.2 **return** *r * confidence value of previous run*

Fig. 9 Confidence value calculation algorithm

Table 1. Maximum number of BV and average confidence values

Maximum BV	Average Confidence Values	
	BF	DF
200	3.58	2.18
400	27.92	13.10
600	54.91	35.68
800	61.10	36.12
1000	75.84	53.45

A. Dangerous Regions of a BVTT

According to our observation in Section III(B), if two objects collide, their BVTT will have one or more trough regions reflecting the collision situation. Here we define the "more probable trough regions for collision" as the *dangerous regions*. If the time available for the collision check process is limited, we may expect the process to return a partial answer about the probability of being in-collision. In our time-budgeted collision detection method, we have to first identify the dangerous regions and use them as the main index for imposing priorities on the order of visiting its children. In other words, the dangerous regions should be checked first.

There is no definite way to identify the dangerous regions. In our system, we make use of the cut list obtained in the previous run as a basis to compute the dangerous region. The computation is based on the assumption that temporal cohesions exist on the cut list of a BVTT for two consecutive collision checks. We segment the nodes in the cut list into groups and calculate the standard deviation for each group. A group with a standard deviation higher than a threshold is considered as a dangerous region.

B. Confidence Values

Because the time resource available for collision detection is limited in our application, the possibility of being in-collision can only be predicted. We have to guess the collision condition with limited information. We propose to define a *confidence value* based on the percentage of nodes being checked in the dangerous regions. As mentioned earlier, the dangerous regions are identified from the cut list of BVTT in the previous run under the assumption that two adjacent cut lists are similar.

The algorithm for computing the confidence value is shown in Fig. 9. The range of the confidence value is from 0 to 100. If the number of bounding volume tests is fewer than the number of allowable checks, the confidence value is set to 100. Otherwise, we calculate the percentage of nodes in dangerous regions that remain collision-free in this run. Finally, we get the confidence value by multiplying this percentage with the confidence value of the previous run.

We have done some experiments to see the effects of maximal number of allowable checks on the confidence values. As shown in Table 1, the averages of confidence values increase with the maximum number of BV checks. This agrees with what one may expect that we are more confident with the result when we are given more time. Therefore, the confidence value is a good index for how confident we are with the current check based on the percentage of crucial nodes that have been checked.

V. Experimental Result

We have done experiments to verify our assumptions in Section III and suggested a strategy for the time-budgeted collision detection method. We have implemented a probabilistic roadmaps planner (PRM) [14] with a parameterized resolution and a pluggable collision detector. To simplify the implementation, instead of limiting the real time spent in collision detection, we set a limit on the number of allowable bounding volume checks.

A. Experimental Data

The scene in Fig. 1 is also used in this experiment. While some difficult problems require more than 1000 bounding volume checks, we set the maximum number of checks to a nominal value of 500. We sample 50 points in PRM in order to build the roadmap in the learning phrase and then find a path between the user-specified initial and goal configurations in the query phrase. A total of 327679 collision detections are required in this experiment. Table 2 shows the results of using two different rules in branching the children's bounding volumes. Table 2(a) uses the basic rule while Table 2(b) uses the best-first rule with priorities as described in the previous section. In Table 2, both tables show only the cases that have reached the number of allowable bounding volume checks. Out of the 327679 checks, 80464 and 80566 checks have reached the limit in the cases of Table 2(a) and Table 2(b), respectively. In these cases, the "number of collision-free checks" column shows the number of collision checks with collision-free results while "number of in-collision checks" shows the number with in-collision results.

B. Branching Priority

In our implementation, we have used two linked lists to record the depths of the nodes in the cut list and a hash table to trace the checked nodes. The overhead of the linked lists and the hash table can be neglected because the time requirements in accessing these data structures are far less than checking the collisions between the bounding volumes.

Table 2 Confidence values and collision percentages

(a) Results of basic bounding volume check

confidence value	nb. of in-collision checks	nb. of collision-free checks	collision percentage
0 ~ 10	206	28204	0.73%
10 ~ 20	47	5347	0.87%
20 ~ 30	40	5012	0.79%
30 ~ 40	26	5003	0.52%
40 ~ 50	22	4821	0.45%
50 ~ 60	27	4549	0.59%
60 ~ 70	20	5145	0.39%
70 ~ 80	8	5300	0.15%
80 ~ 90	9	5585	0.16%
90 ~ 100	2	4908	0.04%
100	0	6183	0.00%

(b) Results of bounding volume check with priority

confidence value	nb. of in-collision checks	nb. of collision-free checks	collision percentage
0 ~ 10	447	52772	0.84%
10 ~ 20	21	4380	0.48%
20 ~ 30	9	3250	0.28%
30 ~ 40	14	2782	0.50%
40 ~ 50	10	2343	0.42%
50 ~ 60	4	2197	0.18%
60 ~ 70	3	2135	0.14%
70 ~ 80	0	2033	0.00%
80 ~ 90	0	2193	0.00%
90 ~ 100	1	2609	0.04%
100	0	3363	0.00%

We have used PQP as the collision detection algorithm for our experiment but changed the recursion rules for performing time-budgeted collision detection. Since collision often happens in troughs, when we need to break a bounding volume into two, we choose to first branch to the child with a deeper depth of checked nodes in the previous run. However, in order to know which child has the deeper traversal nodes, we have to record the depth of the deepest descendant for each node in BVTT. This value is recorded in each node when the BVTT is traversed in the previous run.

In Table 1, we can see that the average confidence values with the best-first (BF) rule are always higher than the ones with the depth-first (DF) rule. In Table 2, we also show that the branching rule with priority (BF) is more effective than the basic branching rule (DF). In almost every confidence value group, the best-first rule leads to a lower collision percentage than the basic rule. More importantly, in the groups with high confidence values, their confidence values seem to have an apparent relation with collision percentages. The higher the confidence value, the lower the collision percentage. Therefore, if a system has limited time for each collision check, it can continuously watch the trend of the confidence values to predict the probability of collisions. However, the acceptable probability for assuming a collision-free situation will mainly depend on the tolerance of the application on incorrect results.

C. Discussions

In our experiments, we have shown that the confidence values are closely related to the collision percentages, and the branching rule with priority helps to reduce the collision percentages. However, not all cases have collision percentages as low as shown in Table 2. Two parameters may also affect the result and require tuning: the maximum number of allowable checks in BVTT and the resolution of continuous collision detection. The larger the maximum number is or the finer the resolution is, the more accurate but time-consuming the collision detectors would be. Therefore, the best set of parameters that can yield the optimal result is yet to be determined probably with an automatic learning approach.

Moreover, we think that the way that the dangerous regions are identified can be improved with a more effective pattern searching algorithm. In addition, the accuracy of confidence values may be improved if the values are looked up from a statistical table constructed in an off-line manner.

VI. Conclusion

The time-budgeted collision detection method proposed in this paper is aimed to provide an answer to a special type of collision detection problem where the allowable time is limited, minor errors can be tolerated, but the answer should be as confident as possible. Under the assumption of good temporal coherence on consecutive checks along a continuous path, we have defined the concept of dangerous region and confidence value to predict the possibility of collision. We have also proposed a branching rule with priority that allows the branch with higher possibility of collision to be traversed first. Although how to handle collision uncertainty still depends on the application at hand, our method has identified a few indices for the application to use when the available time for each collision check is limited.

References

[1] M. C. Lin and J. F. Canny, “Efficient algorithms for incremental distance computation,” *IEEE Conference on Robotics and Automation*, pp. 1008-1014, 1991.

[2] T. Y. Li and J. S. Chen, "Incremental 3D Collision Detection with Hierarchical Data Structures," *ACM Symposium on Virtual Reality Software and Technology*, pp. 139-144, Nov. 1998.

[3] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast distance queries with rectangular swept sphere volumes," *IEEE Conference on Robotics and Automation*, 2000.

[4] S. Gottschalk, M. Lin, and D. Manocha, “OBBTrees: A hierarchical structure for rapid interference detection,” *ACM Siggraph*, pp. 171-180, 1996.

[5] S. Quinlan, "Efficient distance computation between non-convex objects," *IEEE Conference on Robotics and Automation*, pp. 3324-3329, 1994.

[6] G. V. Bergen, "Efficient collision detection of complex deformable models using AABB trees," *Journal of Graphic Tools*, vol. 2, no. 4, pp. 1-13, 1997.

[7] J. T. Klospowslo, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, “Efficient collision detection using bounding volume hierarchies of k-dops.” *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21-37, 1998.

[8] H. Weghorst, G. Hooper, and D. Greenberg, “Improved computational methods for ray tracing,” *ACM Transactions on Graphics*, pp. 52-69, 1984.

[9] B. Mirtich, “V-Clip: Fast and robust polyhedral collision detection,” *ACM Transactions on Graphics*, vol. 17, no. 3, pp. 177-208, July 1998.

[10] A. Foisy and V. Hayward, "A safe swept volume method for collision detection," *The Sixth Int. Symp. of Robotics Research*, pp. 61-68, Oct. 1993.

[11] S. A. Cameron, "Collision detection by four-dimensional intersection testing," *IEEE Transactions Automatic Control*, vol. 6, pp. 291-302, June 1990.

[12] F. Schwarzer, M. Saha, and J. C. Latombe, “Exact Collision Check of Robot Paths,” *Algorithmic Foundations of Robotics V*, pp. 25-41, 2004.

[13] F. Schwarzer, M. Saha, and J. C. Latombe, “Adaptive Dynamic Collision Check for Single and Multiple Articulated Robots in Complex Environments,” *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 338-353, June 2005.

[14] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions Automatic Control*, vol. 12, no. 4, pp. 566-580, 1996.