Efficient Interference Calculation by Tight Bounding Volumes

Masatake Higashi, Yasuyuki Suzuki, Takeshi Nogawa, Yoichi Sano, and Masakazu Kobayashi

Mechanical Systems Engineering Division, Toyota Technological Institute, 2-12-1, Hisakata, Tempaku-ku, Nagoya, 468-8511 Japan
higashi@toyota-ti.ac.jp

Abstract. We propose a method for efficient calculation of proximity queries for a moving object. The proposed method performs continuous collision detection between two given configurations according to the exact collision checking (ECC) approach which performs distance calculation between two objects. This method obtains efficient results as it employs the concept of clearance bounds and performs approximate distance calculations with a tight fit of bounding volumes. The high efficiency of the method, when applied to robot path planning, is demonstrated through some experiments.

1 Introduction

Interference calculation or collision detection [1] is one of the key technologies employed in computational geometry and related areas such as robotics, computer graphics, virtual environments, and computer-aided design. In geometric calculations, a collision or proximity query reports information about the relative configuration or placement of two objects, and it checks whether two objects overlap in space, or whether their boundaries intersect; furthermore, it computes the minimum Euclidean separation distance between the boundaries of the objects. These queries are necessary in different applications, including robot motion planning, dynamic simulation, haptic rendering, virtual prototyping, interactive walkthroughs, and molecular modeling. Some of the most common algorithms employed for collision detection and separation distance computation use spatial partitioning or bounding volume hierarchies (BVHs). Spatial subdivision is the recursive partitioning of the embedding space, whereas BVHs are based on the recursive partitioning of the primitives of an object.

The cost of performing a proximity query, including collision detection and/or distance computation, is often greater than 90% of the planning time involved in robot motion planning [2]. Due to performance related issues, most of the existing planners use discrete proximity query algorithms and perform queries in several fixed sampled configurations in a given interval. This does not assure that they do not miss any thin objects between sampled configurations. Therefore, a method [3] that employs *exact collision checking* (ECC) and assures no collision has been proposed recently. Redon et al. [4], [5] have proposed a different approach; this method checks the collisions between the swept volume of a robot and its obstacles, and achieves a runtime performance roughly comparable to that of the ECC method.

In this paper, we propose a method that employs the concept of ECC but still obtains efficient results, by adopting the following principles.

1. We do not calculate the exact minimum distance between an object and its obstacles. However, we obtain the approximate minimum distance between an object and its obstacles by using BVHs.
2. We avoid calculations if the distance between BVHs is larger than a *clearance bound* that assures no collision by the object movement.
3. We calculate BVHs of obstacles by dividing the bounding volumes according to *approximate convex decomposition* (ACD) algorithm to enhance the tightness of fit and perform the collision tests with a less number of BVHs.

The rest of the paper is organized as follows. Section 2 introduces the clearance bound calculation after the explaining for the ECC. Section 3 proposes a method for obtaining tighter fit of BVHs. Section 4 describes the results of the experiments along with the discussion. Section 5 summarizes the research.

2 Exact Collision Checking and Clearance Bound Calculation

2.1 Exact Collision Checking

Schwarzer et al. [3] introduced an algorithm that executes ECC by using the distances between an object and its obstacles. The object does not collide with obstacles if (1) holds for two configurations q and q'.

$$\rho \cdot d(q, q') < \eta(q) + \eta(q') \ . \tag{1}$$

Here, $d(q, q')$ denotes a path length in the configuration space $\mathcal{C}$. ρ is a space conversion factor for conversion from the configuration space to the work space; it is calculated by using the present configuration. Hence, $\rho \cdot d(q, q')$ denotes the maximum trajectory length of the object in the work space, for the movement from q to q' in the configuration space. $\eta(q)$ denotes the Euclidean distance between the object and its obstacles in configuration q, as shown in Fig. 1. Equation (1) indicates that an object can move freely if the sum of distances from the object to the obstacles is larger than its moving distance. When (1) is not satisfied, we insert a mid point q_{mid} and calculate the distances recursively for $q_{\mathrm{mid}} \in \mathcal{C}_{\mathrm{free}}$; we return 'collision' for $q_{\mathrm{mid}} \in \mathcal{C}_{\mathrm{obstacle}}$.

2.2 Distance Calculation by a Clearance Bound

The cost of performing the proximity query is given in [6]:

$$T = N_{bv} \times C_{bv} + N_p \times C_p \ , \tag{2}$$

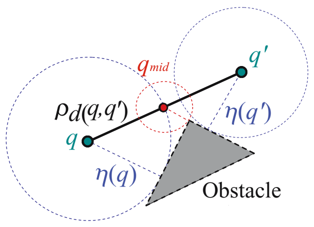


Fig. 1. Exact collision check using distances

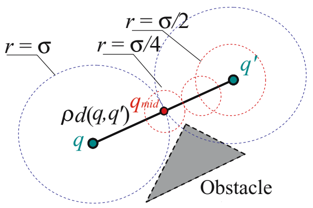


Fig. 2. Collision check by clearance bound

where T denotes the total cost function for proximity queries; N_{bv}, the number of bounding volume pair operations; and C_{bv}, the total cost of a BV pair operation, including the cost of transforming each BV for use in a given configuration of the models and other per BV-operation overhead. N_p denotes the number of primitive pairs tested for proximity, and C_p denotes the cost of testing a pair of primitives for proximity (e.g., overlaps or distance computation).

The computation cost increases significantly if the objects consist of so many points (triangles). In particular, for the distance calculation, the latter part, $N_p \times C_p$, is relatively large, and we must calculate the distance between all the objects searching for pairs of vertices, edges, or planes on them. We do not require a precise distance value, but we must determine whether the object can move from the start configuration to the goal one without colliding with objects.

Therefore, Schwarzer et al. introduced *greedy distance calculation* to compute lower distance bounds in [3]. In addition, we introduce a *clearance bound*. A clearance bound is a distance which is sufficient for an object to clear obstacles. We verify whether the distance between the object and its obstacles is greater than the clearance bound, instead of obtaining the minimum distance. The calculation is executed by using bounding volumes of the objects. If the distance is larger than the clearance bound in the calculations involved in the trees of the bounding volumes, we can stop the calculation at a higher level of the tree. We need not trace the tree to leaves, but we can discontinue the calculation at the corresponding level.

We apply the clearance bound to calculations in ECC. By setting the clearance bound to half the distance in the two configurations: $\sigma = 1/2 \cdot \rho \cdot d(q, q')$, we verify whether $\eta(q) > \sigma$ at both the ends of the interval and cull the distance calculations for cases in which the object sufficiently clears the obstacles. When the object is near an obstacle, the above condition is not satisfied; hence, mid points are inserted until the distance decreases below a given resolution or the robot is in $\mathcal{C}_{\text{obstacle}}$. In Fig. 2, a clearance bound is applied. When $\eta(q') < \sigma$ at q', a mid point is inserted and $\eta(q') > \sigma/2$ and $\eta(q_{\text{mid}}) > \sigma/2$ are verified. In this example, the check is completed at the second level of dividing the interval.

We can also apply the clearance bounds to distance calculations for determining an adaptive step size of the grid-based approach such as BLS [7]. When a global planner wants to select a step size according to the space in a configuration, it sets the given step size as a clearance bound. The robot is collision free if it advances within the clearance bound. Thus, both the step-size determination and ECC calculation are executed simultaneously.

We employ a collision checker called proximity query package (PQP) [8], which pre-computes a bounding hierarchical representation of each object by using two types of bounding volumes: oriented-bounding boxes (OBBs) [9] and rectangle swept spheres (RSSs). OBBs are efficient for detecting collisions of objects, and RSSs are effective for the distance calculations. We execute clearance bound calculation by using the approximate separation-distance-computation, which is provided by the PQP as a function.

3 Tighter Fit of Bounding Volume Hierarchies

3.1 Approximate Convex Decomposition of OBBs

To create a BVH for OBBs or RSSs in PQP, an intermediate OBB is divided into two parts according to the principal directions of covariance matrix (PDC) of the distribution of the input data. However, for obtaining a tighter fit of OBBs, we should divide an intermediate OBB according to the position where it changes direction sharply or at the deepest point of a concavity. Hence, we utilize the approximate convex decomposition (ACD) algorithm [10] for generating an OBB hierarchy. First, we generate a convex hull for the input point data; then, we search the notch point which is not located on the hull and is furthest away from the hull. Finally, we divide the volume into two boxes by the plane passing through the point. The direction of the plane is determined, for example, to equally divide the included angle at the notch vertex. This is repeated until the distance from the point to the hull is below the specified tolerance.

We provide an example of collision checking for two types of solids: C-type and S-type. Figure 3 shows the manner in which each type of solid is divided into an OBB hierarchy. The lefthand-side and righthand-side images show the decompositions by PDC and ACD, respectively. The tightness of fitting to OBBs is measured by the volumes of bounding boxes at each level (see Table 1). A lower hierarchy level indicates a larger volume difference. At the second level, decomposition by ACD is tighter by 13% as compared to that by PDC.

For comparing the computation time required for the two types of decomposed BVHs, collision check is executed for the movement of the left (green) object, as shown in Fig. 4. The result is shown in Table 2, which also includes the number of collision calculations. The computation time is reduced by approximately 40% because a tighter fit separates objects earlier (at a higher level).

3.2 Decomposition for Open Shells

Obstacles are not necessarily composed of complete solids that have no boundaries, but they are sometimes represented by open shells. For example, a car body consists of sheet metal parts; hence it might be composed of open shells.

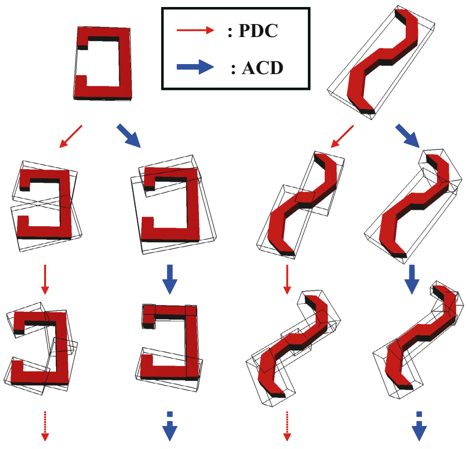


Fig. 3. OBBs by PDC and ACD

Table 1. Comparison of volume of BVHs at each hierarchy

Object	Hierarchy level	Volume	
		PDC	ACD
C-type	0	35.9	35.9
	1	47.5	47.0
	2	37.8	32.7
S-type	0	40.0	40.0
	1	35.2	35.0
	2	32.2	28.3

Table 2. Comparison of computation time for different BVHs

Object		PDC	ACD
S-type	number	30	28
	time (ms)	0.58	0.37
C-type	number	52	46
	time (ms)	1.24	0.74

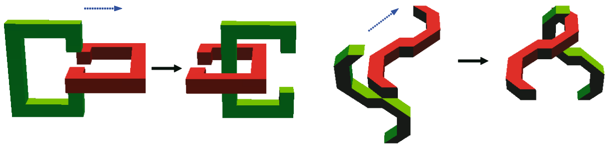


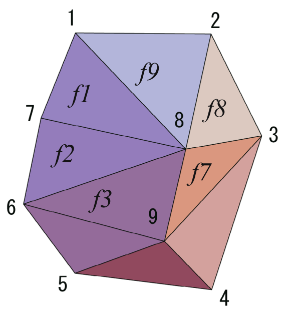
Fig. 4. Movement of objects for collision check

For an open shell, we require a different algorithm of decomposition as compared to that required for a solid object. Since an open shell has boundaries, a point on the boundary might be the most concave part, although it is located on the convex hull. Hence, in addition to the convex hull calculation, we must verify the distance between the concave vertex on the boundary and the convex boundary.

We create a data structure of a shell model from the input data, which include point data with indices and coordinate values and face data with a sequence of vertices expressed by point indices such as mesh data [11]. The input data are stored in two tables, as shown in Fig. 5, in the form of a data structure. From these tables, we form a table of vertex-cycle, which is a sequence of counter-clockwise faces around a centered vertex. The list can be generated according to the following procedure.

Algorithm 1: Generation of point-based data structure (Vertex cycle)

```
   For each vertex ($v_i$),
1) Collect faces from Faces Table, which include $v_i$. Select one face, for example
   $f_p$, and its edge ($v_j, v_i$).
```



INPUT

[Vertices Table]	[Faces Table]
Number of pts	Number of faces
1: x_1, y_1, z_1	1: (1, 7, 8)
2: x_2, y_2, z_2	2: (8, 7, 6)
3: x_3, y_3, z_3	3: (9, 6, 5)
:	:

Generated Data Structure

[Vertex cycle]	[Boundary / Branch edge]
1: (*f*1, *f*9,-)	(1,7,6,5,4,3,2,1)
2: (*f*9, *f*8,-)	
:	
8: (*f*1, *f*2, *f*3, *f*7, *f*8, *f*9)	

Fig. 5. Point-base data structure

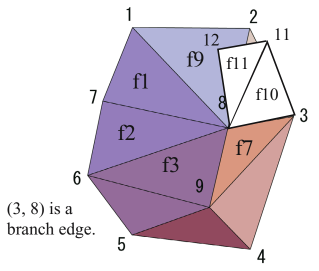


Fig. 6. Shell model with branch boundary

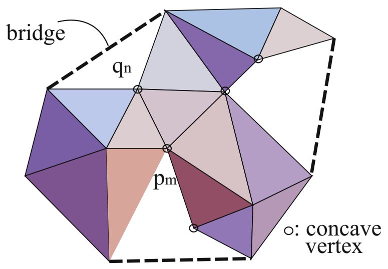


Fig. 7. Convex boundary and bridges

2) Search face f_q, which has edge (v_i, v_j), and store f_q in the vertex-cycle list. If there is no face that includes (v_i, v_j), insert "-"; this means no face and implies that it is a boundary. Then, search the vertex cycle in reverse.
3) Get edge (v_k, v_i) in f_q and set it as the next edge for step 2); repeat steps 2) and 3) until the selected face returns to the first face or the boundary again.

In step 2, if the number of edges is greater than one, it is a *branch edge*, where more than two faces meet. We consider a branch edge as a semi-boundary that separates a shell as a boundary edge. We insert "#" and search faces in the reverse direction, similarly to the boundary edge. As a result, we can obtain multiple vertex cycles.

From the table of vertex cycles, we deduce the boundaries and branch boundaries of a shell model, according to the following steps.
Algorithm 2: Extraction of boundaries

1) Collect vertices from the vertex-cycle table, which include a boundary edge, for example, $(f_i, f_j$,-). Set vertex v_i in the boundary list and set f_i which is next to the boundary, that is "-", as a boundary face.
2) Search vertex v_j, which has a sequence of $(f_i$,-) in the vertex cycle, and add it to extend the boundary list.

3) Select face f_k next to "-" in the vertex cycle of v_j. Replace f_i with f_k and repeat step 2) until we reach the first vertex.
4) Repeat this until there remain no vertices in Step 1. If there are holes, we have multiple boundaries.

We can obtain the branch boundaries by using the same algorithm provided above except for the following conditions: in a vertex cycle, there are multiple pairs of same edges and there may exist pairs of a branch edge and boundary edge. Figure 6 shows an example of a boundary edge. Here, an open shell is added to the shape in Fig. 5. Edge (3, 8) is a branch edge. The vertex cycles for vertex 8 are $(f1, f2, f3, f7, \#, f8, f9)$ and $(\#, f10, f11,$ -). Thus we simply express a non-manifold surface model instead of exact representation [12].

Now, we describe a method to divide an OBB to generate a tight fit of BV hierarchy. First, we divide an object into independent shells according to the obtained boundaries and compose a binary tree for the open shells according to the positions of the shell boundaries. Then, we divide each OBB as a shell, as follows. Here, we define a concave vertex locally for every three points along a three dimensional boundary and compose a concave part of pairs of a *bridge* and concave vertices. A bridge [10] is an edge inserted to connect a concave part and generate a convex boundary. Figure 7 shows an example of a convex boundary and bridges for an open shell. The small circles show concave vertices. The concave vertex that is furthest from the corresponding bridge is the point for division of the shell if the distance is larger than a given tolerance.

The shell is divided along the edges that connect the division point and the nearest vertex on the opposite boundary or the inner boundaries of holes. This division is repeated until there are no concave vertices whose distances to the bridge are greater than the tolerance. In Fig. 7, p_m is a division point and $(\mathrm{p}_m, \mathrm{q}_n)$ is a dividing edge. When there are no concave parts and there exists a hole, we divide its boundary at the extremum vertices along the principal direction of the hole boundary.

4 Experiments

We implemented our local planner for application to rapidly-exploring random trees (RRT) [13] and bi-directional local search (BLS) [7] algorithms. We used the Motion Strategy Library (MSL) [14] to implement and modify RRT as well as GUI for BLS. We show the effectiveness of our planner, applied to RRT and BLS, by conducting some experiments for an object movement in a 2D maze and 3D environments along with an articulated robot in 3D environments. The planner was implemented in C++ and the experiments were executed on a Windows PC (2.4 GHz Pentium 4 processor with 1.0 GB memory).

First, we report the analysis of the effectiveness of our algorithms. Figure 8 shows the start and goal configurations (small red circles) for a moving object in a 2D maze, and Fig. 9 shows those for a 3D cage. We ran RRT 10 times for the 2D maze and 3D cage. Each case involves three types of calculation methods – without ECC, with ECC, and with ECC along with clearance bounds (CB)

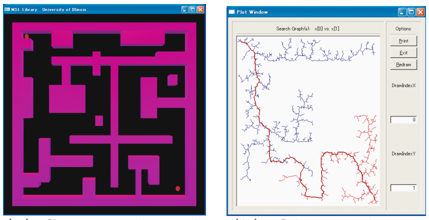

(a) Start and goal configurations (b) Obtained tree and path

Fig. 8. Start and goal configurations for 2D maze

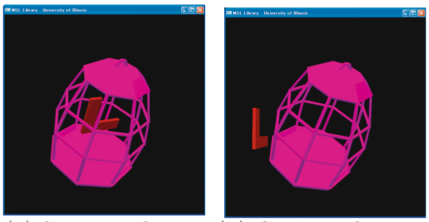

(a) Start configuration (b) Goal configuration

Fig. 9. Start and goal configurations for cage data

Table 3. Experiment results for moving object

		Maze	Cage
CPU Time	Without ECC	28.94	3.71
	ECC	35.90	22.05
	ECC with CB	27.32	11.85
No. of	Collision check	7023	26383
	Nodes of tree	2022	470
	Nodes in path	247	30

Table 4. Experiment result for articulated robot

		Case A		Case B	
		RRT	BLS	RRT	BLS
CPU Time	Without ECC	369.74	42.34	55.87	9.66
	ECC	523.59	145.32	238.25	21.44
	ECC with CB	422.60	115.22	69.84	2.87
No. of	Collision check	16878	90817	7548	1750
	Nodes of tree	5869	4546	2267	467
	Nodes in path	478	2345	110	366

– in the distance calculations. Table 3 shows the results of the calculation. The averages of running CPU time are shown for these methods. The number of points for collision check in the configuration space, number of nodes in the tree, and number of nodes used for the path are shown for ECC with CB. The running time for ECC with CB as compared to that for ECC without CB is reduced by 25%, from 35.9 to 27.3, for the 2D maze. For the cage, the running time is significantly reduced, from 22.1 to 11.9 (approximately half). The difference between the running time in the two cases is due to the following reasons. First, the number of generated nodes in the maze is considerably larger than that in the cage; hence, a large amount of time is required for the tree generation, which is shown in Fig. 8 (b). Second, the time required for 2D distance calculation in ECC is considerably smaller than that required for 3D distance calculation.

Next, we executed experiments on articulated robots that have many degrees of freedom and require a large number of interference calculations. Figure 10 shows the start and goal configurations of the robot along with the given obstacles. The robot is a 6-axes articulated robot and has a fixed base. The surfaces of the robot are represented by 989 triangles and those of a car are represented by 2,069 triangles. In Case A: figures (a) and (b), the robot goes into a car from the outside. However, the robot turns between two cars in Case B: figures (c) and (d). We ran RRT and BLS 20 times for Case B and 5 times for Case A.

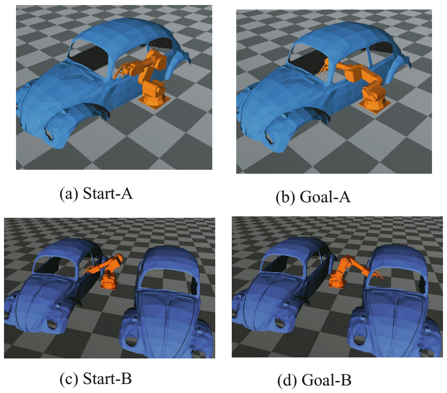


Fig. 10. Start and goal configurations for articulated robot

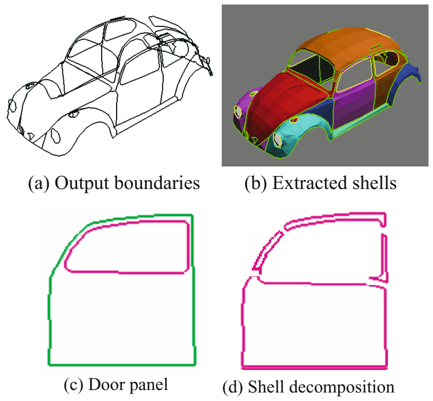


Fig. 11. Boundaries and extracted shells

Each case involves three types of calculation methods, similar to the previous experiments. Table 4 shows the results of the calculations. The running time of ECC with CB compared to ECC without CB is reduced by 20% for Case A; however, for Case B, the running time is reduced drastically from 238.25 to 69.84 (1/3) for RRT and from 21.44 to 2.87 (1/7) for BLS. This is because the robot cannot be separated easily from the BVH due to its position inside the car. In Case B, the time required for ECC with CB is nearly the same as that required "without ECC" for RRT. Hence, the introduction of CB is proven to be effective. For BLS, the time required by the method without ECC is larger than that required for ECC with CB. This is because in the calculation without ECC, we require distance calculations for determining a step size adaptively; however, in the calculation by ECC with CB, we apply the clearance bound to the adaptive step size along with the calculations for ECC. RRT requires a larger time as compared to BLS, because it generates a tree uniformly for free configurations and requires time to search the nearest node to add it to the tree. On the other hand, BLS attacks a target more directly with less nodes and uses discrete adaptive step sizes. Furthermore, BLS applies a lazy evaluation of ECC.

Next, we conducted the decomposition of OBBs by ACD and path calculation for Case A. The reduction of execution time was only 10%, because the object consists of open shells. Hence, we generated a data structure of a shell model and obtained boundaries and shells, as shown in Fig. 11. An example of open-shell decomposition is also shown for a door panel. By using extracted shells, the CPU time decreased to less than one third of the initial value.

5 Summary

We have introduced a method for efficient calculation of proximity queries for a moving object. Our method can be employed for continuous collision detection between two given configurations according to the ECC approach. It obtains results efficiently by using the concept of a clearance bound and approximate distance calculation with close bounding volumes. To obtain close BVHs, we employed algorithms for the decomposition of OBBs. For a solid object, we decomposed OBBs by using ACD; we generated a data structure for a shell object and detected a dividing point furthest from a bridge in the convex boundary. The high efficiency of the method, when applied to RRT and BLS, is demonstrated by experiments for moving objects and practical articulated robots.

Acknowledgments. This study was partly supported by the High-Tech Research Center for Space Robotics from the Ministry of Education, Sports, Culture, Science and Technology, Japan. The implementation of the prototype system was performed in cooperation with IVIS Inc.

References

1. Hadap, S., Eberle, D.: Collision Detection and Proximity Queries. In: SIGGRAPH 2004 Course Notes (2004)
2. Latombe, J.C.: Robot Motion Planning. Kluwer, Boston (1991)
3. Schwarzer, F., Saha, M., Latombe, J.-C.: Exact collision checking of robot paths. In: Boissonnat, J.D., et al. (eds.) Algorithmic Found. Robot V., pp. 25–42 (2004)
4. Redon, S., Kim, Y.J., Lin, M.C., Manocha, D.: Fast Continuous Collision Detection for Articulated Robot. In: Proc. ACM Symp. Solid Modeling and Application, pp. 1316–1321 (2004)
5. Redon, S., Lin, C.: Practical Local Planning in the Contact Space. In: Proc. IEEE Int. Conf. Robot. Autom., pp. 4200–4205 (2005)
6. Lin, M.C., Manocha, D.: Collision and Proximity Queries. In: Handbook of Discrete and Computational Geometry, 2nd edn., ch. 35, pp. 787–807. Chapman & Hall/CRC (2004)
7. Umetani, S., Kurakake, T., Suzuki, Y., Higashi, M.: A Bi-directional Local Search for Robot Motion Planning Problem with Many Degrees of Freedom. In: The 6th Metaheuristics International Conference MIC 2005, pp. 878–883 (2005)
8. Larsen, E., Gottschalk, S., Lin, M., Manocha, D.: Fast Proximity Queries with Swept Sphere Volumes. Technical report TR99-018. Department of Computer Science, University of N. Carolina, Chapel Hill (1999)
9. Gottschalk, S., Lin, M., Monacha, D.: OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. In: Proceedings of ACM SIGGRAPH 1996, pp. 171–180 (1996)
10. Lien, J.-M.C., Amato, M., Approximate, N.: convex decomposition of polygons. In: Proc. 20th Annual ACM Symp. Comutat. Geom (SoCG), pp. 17–26 (2004)
11. Botsch, M., Pauly, M.: Geometric Modeling Based on Polygonal Meshes. In: SIGGRAPH 2007 Course Notes (2007)
12. Higashi, M., Yatomi, H., Mizutani, Y., Murabata, S.: Unified Geometric Modeling by Non-Manifold Shell Operation. In: Proc. Second Symposium on Solid Modeling and Applications, pp. 75–84. ACM Press, New York (1993)
13. Kuffner, J.J., LaValle, S.M.: RRT-connect: An efficient approach to single-query path planning. In: Proc. IEEE Int. Conf. Robot. Autom., pp. 995–1001 (2000)
14. LaValle, S.M.: Motion Strategy Library, `http://msl.cs.uiuc.edu/msl/`